2 mif

3

HUGHES

HUGHES AIRCRAFT COMPANY
GROUND SYSTEMS GROUP

(NASA-CR-120204) DESIGN OF A MODULAR DIGITAL COMPUTER SYSTEM DRL 4 AND 5 Final and Phase 3 Report (Hughes Aircraft Co.) ~~260~~ 257 p HC $16.00 CSCL 09B

N74-20840

Unclas
G3/08 16492

DESIGN OF A MODULAR DIGITAL COMPUTER SYSTEM DRL 4 AND 5 FINAL AND PHASE III REPORT

CONTRACT NAS8-27926
DECEMBER 21, 1973

DESIGN OF A MODULAR DIGITAL COMPUTER SYSTEM
DRL 4 and 5
FINAL AND PHASE III REPORT

December 21, 1973

Prepared under Contract NAS8-27926

by

HUGHES AIRCRAFT COMPANY
FULLERTON, CALIFORNIA

for

George C. Marshall Space Flight Center
National Aeronautics and Space Administration

FR 73-11-998

FOREWORD

This report documents the accomplishments of contract NAS8-27926, whose scope is the design of an Automatically Reconfigurable Modular Multiprocessor System (ARMMS), with an emphasis placed on the work performed during Phase III of this contract. The contract's time period was from October 8, 1971 to December 31, 1973 with work performed after March 1, 1973 falling under Phase III. The design is being performed by the Data Processing Products Division of Hughes-Fullerton. Hughes Space and Communication Group in El Segundo, California provided support in the area of Aerospace Component and Packaging Technology and M&S Computing, Inc. of Huntsville, Alabama is providing support in the area of executive software design under subcontract to Hughes. The design is being directed by the Astrionics Laboratory of NASA's Marshall Space Flight Center in Huntsville, Alabama. The contracting Officer's Representatives are Dr. J.B. White and Mr. Sherman Jobe.

This report was edited and prepared by R.A. Easton. W.L. Martin headed this project during Phases I and II, R.A. Easton during Phase III. Major individual contributors to this report included R.A. Easton – ARMMS Hardware design; W.L. Martin, W.G. Tees – early ARMMS Hardware tradeoffs; D.W. Kuyper – 10P design; S.A. Simpson, B. Cohen, R. Radys – Component and packaging technology; J.H. Engleman, J.L. Bricker Reliability Data Base and modeling, respectively; and T.T. Schansman, K.H. Schonrock, C.E. Turner, D.J. Hyde – ARMMS Software.

CONTENTS

SECTION 1

DESCRIPTION OF CONTRACTUAL WORK REQUIREMENTS

The scope of the work requirements for contract NAS8-27926 as contained in its Statement of Work are as follows:

The contractor, utilizing as much of the Spacecraft Ultrareliable Modular Computer (SUMC) detailed logic as possible, shall design a modular digital computer system for space flight applications. The design shall entail not only system engineering for the total computer system, but shall also include detailed design for the memory, system controller (BOSS), input/output unit, and error detection, isolation, and switching mechanism necessary for the application of redundancy. The computer system shall be capable of operating in three basic modes.

1. Internally redundant mode to provide relatively low computational capability but a very high reliability.

2. Parallel processing mode such that parallel CPE's can handle different computational tasks. This provides a large amount of computational capacity with a relatively low reliability requirement.

3. The system must be capable of operating when at least one module of a kind (i.e., one of n modules in any or all redundant stages) is functional.

The intent is to provide a system which can be used in a wide variety of applications. First, the system must be capable of operating as an internally redundant system for periods of time when real time recovery from failures is required; e.g., in the launch phase of a vehicle. That is, failure must be detected, isolated, and masked or corrected without resorting to special purpose diagnostic software. The level of modularity and the degree or amount of redundancy shall be dictated by the reliability specification herein. Second, the system must be capable of operating as independent parallel modular processors during periods of time when very high computational capabilities are required. Thus, the tasks performed by each processor, although possibly dedicated, are different. The approach may require the design of a so-called BOSS executive controller module. An example of this application may be in a large space station which requires a multitude of varied computational requirements. Third, the system must be capable of operating as a simplex system when at least one module of a kind is operational to provide a high probability, 0.99, of having at least one operating processor at the end of a five-year mission. This allows not only some computer capacity at the end of a long mission, but also provides for degradation; i.e., some tradeoffs between computational capacity and reliability are provided. e.g., a mission to the outer planets, such as the Grand Tour or planetary softlandings. The basic objective is to provide a system with extremely high reliability, very large computational capability, or a system where these can be traded off. The last item is sometimes referred to as "graceful degradation."

The Central Processor Element (CPE) shall be assumed to be based on the MSFC SUMC design. The contractor shall examine this design and define the modular partitioning required to meet the system requirement. The design of the memory system, input/output, executive controller and failure detection, isolation and switching logic shall be performed by the contractor and integrated into the overall system. The input/output unit shall be a standard type with one input and output channel interfacing directly to memory, i.e., the CPU is not to be burdened with the total input/output problem. The input/output unit will interface the computer system to a single device which for purposes herein will be assumed to be a data bus system. It is to be assumed that the data bus system can accommodate serial information at a peak bit rate of 10 MHz.

Special attention is to be given to partitioning the system in an optimum manner so that parallel redundancy can be applied to each portion. In partitioning, basic consideration must be given to the number of interconnection between units, reliability, etc. Parallel standby modules are to be assumed to be in a powered-off mode. Particular attention must be given to solving the problems of failure detection, failure isolation and module switching. Module switching is necessary not only in switching out failed modules and switching in standby units, but also in transferring from a parallel redundant mode to a simplex parallel processing mode. The system must be capable of detecting intermittent as well as solid failures in all three modes of operation. In the first mode of operation, using modular redundancy, the error correction must be in real time. This infers special purpose hardware for error detection and module switching. In the second and third modes of operation, the time required for error correction must be held to a minimum. Thus, in these modes, special purpose hardware or diagnostic software may be employed. The contractor shall perform system design to the functional level. The contractor must show and demonstrate that he has solved all problems associated with error detection and correction. In some cases, detailed logic may be suitable whereas in others demonstrational models or breadboards may be required.

The contractor shall design the executive software system insofar as it is required to participate in the overall system design for accomplishing failure detection and failure correction. The software design shall be detailed to the level necessary to begin implementation. Flow charts must be provided as part of the documentation for the software design. It shall be assumed that the tasks to be performed by the system are typical guidance and navigation problems during launch and interplanetary missions as well as providing data management. The requirements for the executive software system, as well as the hardware for the executive controller, in the areas of failure detection, failure correction, system reconfiguration, and system verification are to be defined by the contractor. Any special instructions required to aid in fault isolation must be identified. Design verification and support software plans for the above software must also be developed.

The contractor shall develop all mathematical and computer models necessary to carry out the research herein. A reliability model incorporating consideration for failure detection and correction shall be developed and used to determine if the requirements specified herein have been accomplished. The relative complexity of the system when compared to a simplex system shall be determined. Computing capacity, reliability and degradation shall be analytically defined such that tradeoffs can be made in these parameters.

The above scope of work shall be accomplished in three basic phases: Phase I shall be the selection and definition of the configuration which satisfies the requirements for the five-year mission. This shall include partitioning of MSFC's CPE, preliminary design or selection and partitioning of a memory, input/output unit, and executive controller. In other words, a simplex system will have been defined, and a preliminary design at a functional level completed and partitioned so that it can be made redundant. Phase II will entail incorporating redundancy into the design. Extensive consideration will be given to the problem of failure detection and correction both in determining what is required as well as defining how it will be implemented. Detailed logic design of the decision element is required and possibly breadboards to demonstrate feasibility. Reliability models incorporating the decision element will be developed and analyzed to determine if the desired goals are being achieved. The degree or amount of redundancy to meet the requirements will also be determined. Phase III will consist of the next level of design detail and a more detailed analysis of the system. Detailed design to the logic level may be required in problem areas. Repartitioning of the system may be required to improve reliability or otherwise enhance the design of the system. The mathematical or computational models will be modified to take into consideration more design details.

Further definition of the BOSS and CPE modules during Phase III is of primary importance in ARMMS. First, like the switching elements, their unique characteristics cannot be directly extrapolated from earlier computer experience. Second, they play as fundamental a role in achieving the reliability objectives as do the switches. Specific features which shall be investigated further include but are not necessarily limited to the following:

1. Redundancy incorporation to achieve the reliability objectives.
2. Detailed methods of controlling the switches.
3. Translation or tradeoffs of software requirement into hardware requirement.
4. Identification of the role of BOSS in system synchronization.
5. Investigation of hardware means to improve overall system efficiency.
6. Investigate commonality of BOSS elements with other processing elements.
7. Generate BOSS system definition and specifications in relation to the other elements in the system.
8. Perform evaluations of the applicability of ARMMS Fault Tolerance Techniques to a SUMC processor using the existing LSI module set and of these modules to the ARMMS CPE.
9. Perform a high reliability system design (exclusive of BOSS), including the logic design of a "mini-BOSS" module that will serve as the system's high reliability switching core.
10. Perform detailed logic design of BOSS and/or CPE error detection and masking logic.

Consistent with the related results of the system design (number of intermodule connections, gate count estimates, etc.) and the expected packaging environment, concepts for packaging and assembling ARMMS will be evolved. Each mode of operation shall be investigated and a system efficiently adaptable to all these modes shall be developed. Estimates of total power, weight, and volume for the range of configurations shall be made assuming LSI implementation. The estimates shall be based on one or more specific technologies. The impact of minimum versus maximum power circuit technologies shall be described. Problems and risk areas, if any, shall be identified. Artistic drawings for one or more concepts shall be delivered to MSFC. Power, weight and volume of the total system shall be minimized for each type of mission. The range of environmental constraints (temperature, vibration, vehicle form factors, etc.) encountered in boost, orbital, lunar, and interplanetary missions must be met.

SECTION 2

SUMMARY OF ACCOMPLISHMENTS DURING THE ARMMS PROGRAM

The primary objective of contract NAS8-27926 is to perform the system design of an advanced modular computer system designated the Automatically Reconfigurable Modular Multiprocessor System (ARMMS). The effort to be described is fully compliant with the scope of work as given in the previous section.

Any computer system justifies the cost of its development to the degree that it provides new capabilities or allows earlier ones to be satisfied at reduced cost. ARMMS is primarily oriented toward providing the following new capabilities for spaceborne computers for application in the 1975 to 1985 time period:

1. To provide a modular computer system which is responsive to many mission types and phases.
2. To achieve through modularity a higher computing capability than previously available for spaceborne application.
3. To provide the capability to choose to maximize reliability through the use of redundancy or to maximize processing capacity through multiprocessing. Moreover, this multi-mode capability must be dynamic; that is, a given system may alternate from one mode to another as a function of real-time requirements.
4. To maximize reliability in all applications through the incorporation of fault detection and recovery features and through the use of high reliability components.

The first consideration of any ARMMS design tradeoff has been to avoid compromising these basic objectives. However, an advanced paper design will surely remain only that unless continuous concern is maintained for the practical requirements of implementation. Such design parameters as power density, weight, volume, pin count, device count, etc., must influence the design process. The design as presented here is oriented toward achieving the ARMMS objectives within a practical hardware and software context.

ARMMS is an outgrowth and extension of two NASA development programs, the MSFC Space Ultrareliable Modular Computer (SUMC) and the ERC Modular Computer. The SUMC program has emphasized the development of a processor which is effectively partitioned for LSI implementation. To date, a breadboard TTL prototype has been constructed and a MOS LSI version is nearing completion. A modified version of SUMC is anticipated to be the processor module of the ARMMS system. The breadboard of the ERC Modular Computer which has undergone evaluation at MSFC had the common objective with ARMMS of achieving a variable configuration for varying levels of processing capacity and reliability.

In addition, the experience of numerous NASA, Air Force, Army and Navy architecture and design studies have been reviewed and incorporated into the ARMMS design where appropriate. In general, these efforts have considered a subset of the ARMMS objectives. For example, the JPL STAR is oriented toward long-life reliability. The MSC reconfigurable guidance and control computer study considers primarily space shuttle requirements. Other studies have considered space station computer requirements. All have identified design principles which form a substantial base of experience for the ARMMS development.

The 27-month contract has been divided into three phases. The program plan as performed during these Phases is shown in Figure 1. At the inception of the contract, an initial baseline description was provided by MSFC. The primary effort in Phase I was to establish general design guidelines necessary to achieve the ARMMS reliability and performance objectives; to survey published estimates of performance requirements for future space computers, and to refine the initial baseline. The efforts during Phase II were aimed at system and interface design including definition of the overall system response to all classes of failures.

Power supply and logic family tradeoff studies and preliminary studies of memory and BOSS module register level design, BOSS/CPE commonality and ARMMS Control Executive Software (ACES) were also completed. During Phase III final versions of the register level designs for all ARMMS module types were completed. In addition, applicability of the SUMC LSI module set to ARMMS was evaluated, a feasibility study of a BOSS-less version of ARMMS was performed and studies of ARMMS reliability modeling, ARMMS packaging, and ARMMS support and control executive software including memory utilization estimates and a design verification plan were completed. A summary of work performed during phases I and II and a detailed description of Phase III work is contained in the remaining sections of this report. The general subject of each is listed below:

SECTION 3 – ARMMS SOFTWARE DESIGN

This section begins with a summary of ACES: ARMMS Control Executive System covering software philosophy, task control, event recognition and response, resource allocation and control, fault detection and diagnostic processing, information protection, and input/output control. The following topics describe three additional software studies performed covering ACES timing and memory utilization estimates, ARMMS support software requirements, and an ACES Design Verification Plan. All software work on this contract was performed by M&S Computing, Inc. under subcontract to Hughes.

SECTION 4 – ARMMS HARDWARE DESIGN

This section begins with a summary of hardware design tradeoffs and guiding assumptions made prior to phase III effecting the final ARMMS design. These include choice of operating modes, executive function location, module partitioning, memory hierarchy, fault tolerance approach, and configuration architecture. Register level designs and reliability analyses based upon these designs are given for each ARMMS module in the next topics. The final three topics cover tradeoffs requested by MSFC in order to bring ARMMS closer to the requirements of present SUMC related programs and known near-term missions to which ARMMS is believed to be applicable. The first describes modifications to SUMC to allow its use as an ARMMS CPE. The second describes a BOSS-less version of ARMMS for missions not able to afford or justify a full ARMMS system. The last summarizes the technical aspects of an ARMS

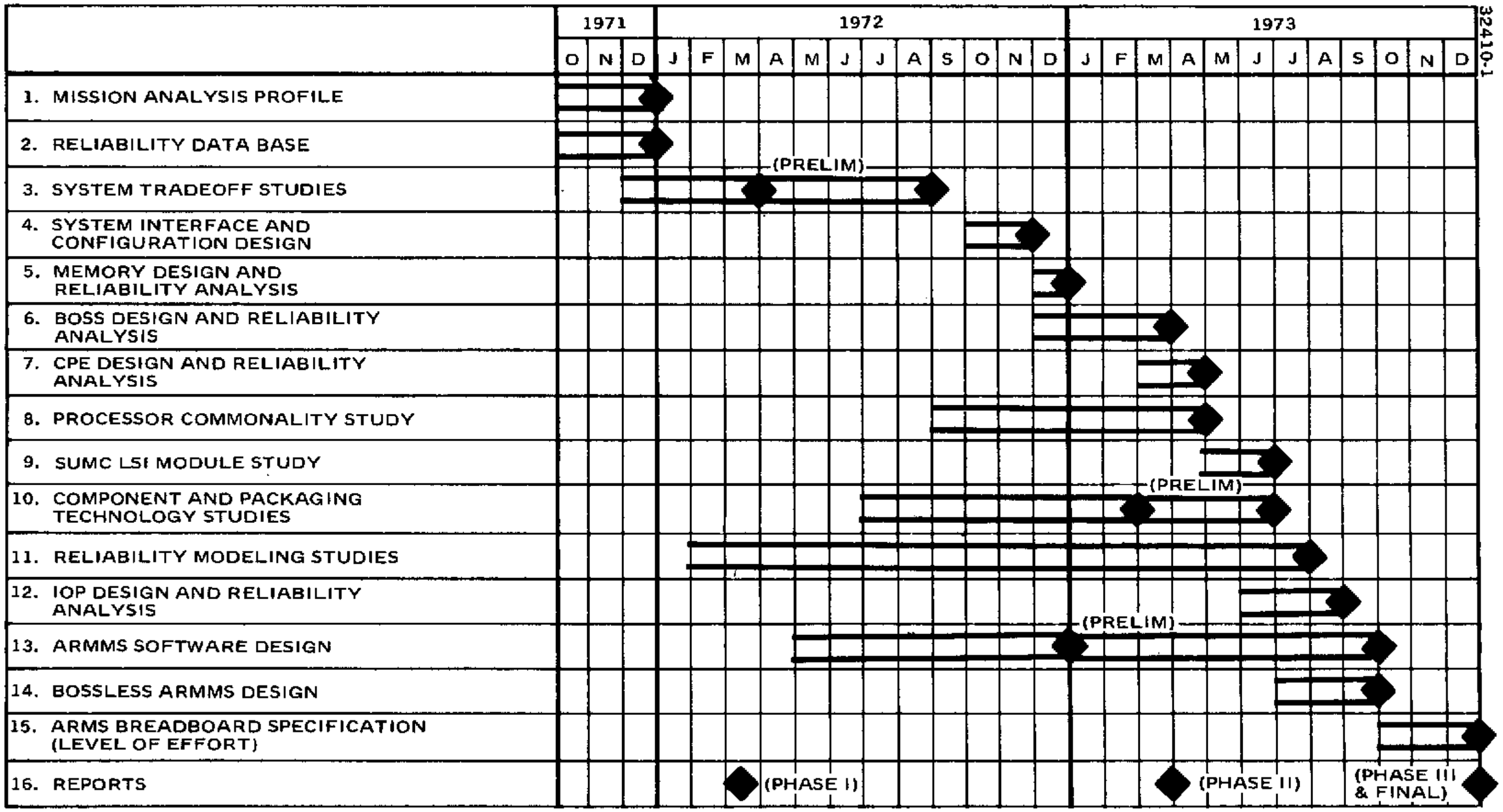


Figure 1. ARMMS Design Plan

(ARMMS with no multiprocessing capabilities) breadboard based on ARMMS principles modified as described in these previous two subsections. The breadboard will be implemented at Hughes during 1974.

SECTION 5 – ARMMS COMPONENT AND PACKAGING TECHNOLOGY STUDIES

This section consists of two parts. The first summarizes the component technology tradeoff studies performed during Phases I and II in the areas of data bus technology, logic families, and power supply configurations. The second gives the results of a study to define packaging concepts and physical hardware parameters for each of the ARMMS module types and for a range of typical ARMMS configurations. Areas investigated included LSI chip and discrete component packaging methods, printed circuit board and chassis design, module interconnection techniques, and thermal and stress analysis of the design chosen.

SECTION 6 – ARMMS RELIABILITY STUDIES

The first part of this section summarizes the reliability data base study performed during phase I which yielded the failure rate numbers used in the module reliability analyses discussed in section 4 of this report. Equations for hand calculating ARMMS reliability using the numbers from section 4 are also given. The final topic surveys reliability studies performed elsewhere, assessing their degree of applicability to ARMMS, and then describes a new model developed specifically for ARMMS.

SECTION 3

ARMMS CONTROL EXECUTIVE SOFTWARE

This section discusses ACES: ARMMS Control Executive System, the software design effort which was performed in close coordination with the ARMMS hardware design to insure a soundly integrated design of the system. First the objectives and scope of ACES are outlined. Subsequently the Control Executive Concepts, from a users viewpoint, and detailed design information are presented. Software philosophy, job and task control, event processing and recognition, resource allocation and control, fault detection and diagnostic processing, information protection, and input/output control concepts are covered.

To insure that all major software problems had been considered, three special studies were performed: ACES timing and memory utilization estimates were made, potential ACES Design verification methods were reviewed and recommendations made, and ARMMS support software requirements were investigated in detail. Recommendations were made concerning the types of support software required and potential use of existing packages. The results of these last three efforts are summarized at the end of this section. All software work on this contract was performed by M&S Computing, Inc. under subcontract to Hughes.

ABBREVIATIONS

ABEND -	Abnormal Ending or Termination
ACES -	ARMMS Control Executive System
AFI -	Alert File Item
AFM -	Alert File Memory
ARMMS -	Automatically Reconfigurable Modular Multiprocessing System
AVAIL -	Available Resource Word
BOSS -	Block Organizer and System Scheduler
BSW -	Bus Status Word
CPE -	Central Processing Element
CSRW -	Configuration Stream Request Word
DIO -	Direct Input/Output
DP -	Diagnostic Processor
FBSM -	File Block Status Matrix
FD -	Fault Detector
FM -	File Memory
FPS -	Full Processing Stream
I/O -	Input/Output
IOP -	Input/Output Processor
IOPS -	I/O Processing Stream
IP -	Input to (CPE) Processor (Bus)
JAL -	Job Active List
JDF -	Job Definition File
JIB -	Job Information Block
LA -	Logical Address
LAAT -	Logical Address Assignment Table
LM -	Logical Module
LP -	Logical Page
LPS -	Limited Processing Stream
LSI -	Large Scale Integration
LU -	Logical Unit
MAXWATE -	Maximum Available Stream Weight
MET -	Master Execution Table
MFW -	Module Fail Word
MI -	Memory Input (Bus)
MIC -	Memory Input (Bus from) CPE

ABBREVIATIONS
(continued)

MINPRI -	Minimum Priority Needed to Pre-empt
MIP -	Memory Input (Bus from) IOP
MO -	Memory Output (Bus)
MOC -	Memory Output (Bus to) CPE
MOP -	Memory Output (Bus to) IOP
MSW -	Module Status Word
OB -	Output Bus (IOP to VS)
PEQP -	Priority Execution Queue Pointer
PLIST -	Priority Execution List
PO -	(CPE) Processor Output (Bus)
PSW -	Program Status Word
Q -	Queue (Timer Queue or Priority Queue)
RERQ -	Resource Requirements Table
RPC -	Resource Pool Counters
TD -	Task Dictionary
TDIB -	Task Dictionary Information Block
TDIF -	TMR Dispatcher Inhibit Flag
TMR -	Triple Modular Redundancy
TQI -	Task Queue Item
TQM -	Task Queue Memory
TTE -	Time to Execute
UST -	Unit Status Table
VS -	Voter Switch
WF -	Weighting Factor
WFM -	Wait File Memory
WFP -	Wait File Pointer
WI -	Wait Item
WIQ -	Wait Item Queue

3. ARMMS CONTROL EXECUTIVE SYSTEM (ACES)

This section describes the software design effort performed in support of ARMMS. The effort was performed in close coordination with the ARMMS hardware design to insure a soundly integrated design of the system.

The major part of the effort was directed towards the development of the Control Executive. This section, therefore, first describes the objectives and scope of the system. Subsequently the Control Executive Concepts, from a users viewpoint, and detailed design information are presented.

To insure that all major software problems had been considered, two special studies were performed. Potential methods for design verification were reviewed and recommendations made.

Support software required for ARMMS application and Control Executive implementation was investigated in detail. Recommendations were made concerning the type of software packages required and potential use of existing packages. The results of these last two efforts are summarized at the end of this section.

3.1 Control Executive System Design Objectives

A primary objective of ARMMS is to provide the ability to support a long life mission with a high probability of success. ARMMS can therefore, for example, be configured as a TMR System with standby spares for each module.

ACES, therefore, must first of all be able to react to error indications from the hardware, isolate a failing module, switch in a spare module, and allow the system to continue successfully. This has to be accomplished without any human assistance. ACES must further be able to allow the systems to degrade gracefully until the point that all of a particular type of module have failed. In addition, ACES must provide the application designers with as many aids as possible to prevent the propagation of software errors. That is, the effect of undetected software bugs must be contained within the software module containing the error. This may allow the system, in most instances, to continue its most critical functions regardless of software failures.

ARMMS can be selected to be configured as a high-performance system consisting of modules identical to those used in the high reliability mode described above. To accomplish this the system can be configured into a multiprocessing system.

ACES must therefore be able to schedule execution of programs on a varying number of independently operating modules. It must allow an application to be designed such that it can be divided in concurrently executing modules. It must not, however, force an application into a special design when multiprocessing is not necessary. Program modules, executing concurrently, must, of course, be prevented from interfering with each other's operation.

The primary types of applications, which ARMMS is anticipated to support, are real-time applications such as vehicle control, experiment control, etc. ACES is, therefore, primarily designed to support "process-control" type applications. This does not imply that "batch-processing" will not or cannot be performed. It implies that many support services characteristic of "batch-processing" (such as File Management) are not a standard service within ACES, but many "real-time control" services are. It is anticipated that, where batch-processing is required, that particular job and its support service routines are run as a single task under control of ACES. Batch-processing is thus considered incidental to the ACES design.

Finally, it is necessary to keep the ACES system as small and simple as possible. ACES directly influences BOSS and its interfaces with the ARMMS modules. The complexity of BOSS and its interfaces directly affect the overall reliability and cost of the system. In addition, the Control Executive itself must not fail, (nearly) exhaustively. The ACES design must, therefore, lend itself to a true modular design; that is, a design with simple interfaces between modules, resulting in a finite number of combinations of inputs and outputs for each module.

3.2 Scope of the Control Executive System

Any software development effort needs to have boundaries established to insure that it fulfills its intended purpose and does not include functions that were not intended to be provided.

The following describes, in outline form, the scope under which the ARMMS Control Executive System (ACES) was developed.

I. Job Management

A. Job Control

The system provides support for the concurrent execution of multiple jobs.

The system allows jobs to be scheduled by other jobs based on real time/time intervals or remote requests.

1. Job Scheduling

a. Job Scheduling Algorithm

Scheduled jobs are selected for activation based on job priority and memory resources available. They are not deactivated until normal or abnormal termination.

b. Job Scheduling Initiation

All job scheduling requests are initiated by tasks (application or system).

c. Job Scheduling Queue Maintenance

The system maintains a job input queue, with a maximum of sixteen (16) entries.

2. Job Resource Allocation

a. Job Core Storage Allocation

Static core storage allocation is provided at the partition level.

Core storage remains allocated until job completion.

The system provides static core allocation for areas common to jobs.

b. Common Routine Allocation

The system supports the inclusion of serially reusable and re-entrant common subroutines.

The system allows user provided routines to be shared among jobs under protection of the executive.

3. Job Loading

The system provides for job loading into main memory from components available in the system library.

Jobs are loaded in an absolute format; unresolved linkages can be resolved at load time.

The system supports a simple job-step structure.

4. Job Termination Processing

The system deallocates all resources at job termination.

The system provides the option to execute a prespecified job at task abnormal termination.

B. Task Control

The system supports the specification and execution, and coordination of asynchronous execution of tasks on multiple processing streams within a job.

1. Task Scheduling

a. Time Initiated Scheduling

The system permits a task to be scheduled at a specified absolute time.

The system permits a task to be scheduled after a specified time interval.

The system permits a task to be scheduled periodically at each elapsement of a specified time interval.

b. Event Initiated Scheduling

The system provides scheduling which is conditional upon recognition of the following events or combinations thereof:

(1) External attention requests

(2) Error conditions

(3) I/O completion

(4) Task completion (normal/abnormal)

(5) Intertask program flags

c. Task Initiated Scheduling

The system provides for task initiated scheduling of:

(1) Jobs

(2) Job phases within the same job

(3) Tasks within the same job

The system provides scheduling for "immediate" execution of other tasks or common subroutines.

The system provides scheduling for asynchronous execution.

The system provides scheduling for subsequent execution.

d. Task Scheduling Queue Maintenance

The system allows a large number of tasks (≤ 100) to be scheduled.

e. Event Synchronization

The system supports a suspension of task execution until recognition of the following events or combinations thereof:

(1) Specified absolute time

(2) Elapsed time interval

(3) External attention request

(4) Error conditions

(5) I/O completions

(6) Task completions (normal/abnormal)

(7) Intertask program flags

2. Resource Allocation

a. Core Storage Allocation

The system provides dynamic core allocation for:

(1) I/O buffers

(2) Work storage

The system provides dynamic read and/or write protection on any area used by a task.

b. I/O Device Allocation

The system permits device specification at the generic device level.

c. Common Routine Allocation

The system supports the use of routines common to tasks within a job of the following types:

(1) Serially reusable

(2) Re-entrant

The common routines are explicitly identified and may reside in the problem program area.

d. Processor Stream Allocation

The system supports the allocation of processing streams on a task level in accordance with predefined parameters specified for each task.

Processing streams are deallocated on any type of task completion

3. Dispatching Control

The system supports dispatching based on preassigned task dispatching priorities and availability of allocatable resources.

The system supports dynamic dispatching to any combination of resources forming a valid processing stream.

4. Task Termination

The system deallocates all task resources upon abnormal task termination.

The system allows a specified task to be executed upon abnormal terminations.

C. I/O Control Interface

The system is able to interface with a variety of I/O processors (and consequently devices) subject to standard interface requirements.

The system is able to support "asynchronous" I/O (task execution does not halt) as well as "synchronous" I/O (task execution suspended until I/O operation complete).

1. I/O Scheduling

The system provides the capability for a task to request execution of an I/O request without suspending execution of the task.

The system provides a means whereby a task can monitor an I/O request's completion without suspending the task's execution.

Specific device assignment is the responsibility of the system.

The system permits the specification of I/O request priorities.

The system provides facilities for alternate I/O (bus or device) routing.

2. Data Transfer

The system provides buffer control.

The system is able to interface with a basic file manager.

a. Buffering Control

The system provides for simple buffering of data.

The system provides dynamic buffering of data.

D. System Communication Interface

The system allows for a command interface to override or invoke its functions concerned with automatic mission scheduling, and reconfiguration.

The system allows for an interface with a possible test (hardware or debug (software) console or loading mechanism.

1. Resource Status Modification

The system allows modification of resource status from an on-line console or command processor (remote control).

2. System Status Interrogation

The status of the system is available through any of the communication interfaces.

The system provides facilities to display the following:

(1) Resource status

(2) Task status

(3) Task information

(4) Queue status

II. Diagnostic Error Processing

The system insures that the effect of errors caused by execution of a task is limited to that task. That is, neither the Control Executive nor other tasks should be affected by the failures in a task.

A. Hardware Error Control

1. Error Correction

The system fully utilizes the reconfiguration capabilities provided in ARMMS to replace failed (or potentially failed) modules with operational (fully or partially) modules.

The system provides control linkage to user (task) abort routines upon detection of conditions that prohibit successful task completion.

The system diagnoses equipment malfunctions at least to a module level.

2. Error Notification

The system logs out errors and takes appropriate actions upon detection of errors.

3. Error Recovery

The system permits on-line system maintenance of devices.

The system allows commanded reconfiguration through any of its system communication channels.

B. Software Error Control

1. Error Correction

The system provides controlled linkage to user error abort routines upon detection of software errors (or potential software errors).

The system provides a default action if no user routines are provided.

The system is able to detect that hardware errors are causing errors that seem to be software errors.

2. Error Notification

The system denotes the fact that a task has been aborted.

C. Interface Error Control

The system dynamically validates all external or internal linkages to the fullest extent possible.

III. Processing Support

A. Timing Service

1. Real Time Clock Service

The system provides the current real time in hours/minutes/seconds.

The system provides facilities for task suspension until a specified time.

2. Interval Timer Service

The system provides one interval timer.

The system permits time intervals to be measured in terms of actual elapsed time.

The system permits task suspension for a specified time interval.

The timer base is fixed.

B. System Test Mode Services

The system provides I/O facilities to reroute I/O requests.

The system allows the user to override abnormal abort services.

The system allows for the insertion of breakpoints in programs.

The system allows the user to start or restart a program at a specified address.

The system permits memory searching/display.

The system permits memory modification.

C. Maintaining Error Statistics

The system accumulates information for a hardware error summary.

The system accumulates information for a software error summary.

The system provides facilities for the analysis of error statistics.

D. Event Monitoring

The system monitors external signals (discretes, interrupts) as well as internal signals (program flags) or requests from the tasks.

The tasks are able to control their execution based upon the status of these events.

The tasks are able to base their decision upon the status of such events.

E. Common Data Access and Protection

The system provides a common data area accessible to all jobs in the system.

The system provides Read or Write locks on groups of variables in any common area on request of a task.

The system prevents deadlocks due to access to common variables.

3.3 Control Executive System Concepts

This section presents the ARMMS Control Executive System as it would be employed by a user. While any system can be technically involved and logically sound within itself, the true test of a "good" operating system lies in its ability to perform many meaningful and beneficial functions for its user(s).

ACES makes available a comprehensive set of over 20 request services to the user. In addition, many useful techniques from larger scale operating systems have been incorporated into ACES design. Such techniques include multitasking, multijobbing, dynamic working storage allocation, etc. Table 3-1 lists the request services provided by ACES to the user.

The following paragraphs summarize the major capabilities provided for the user by ACES. All capabilities and services presented are explained as they would be utilized by the ACES user.

3.3.1 Job Control

In the ARMMS system, a job is the highest user entity processed by ACES. A job is composed of one or more tasks which perform different, but related functions. For instance, in the space environment, for which ARMMS is designed, one job might be for vehicle control, one for a life support system, while another would be for performing experiments. The vehicle control job might contain such tasks as navigation, guidance, minor loop, minor loop support, and switch selector processing. ACES supports a maximum of four jobs in execution simultaneously.

In many cases, all tasks of a job are not necessarily required to be in main memory simultaneously. A particular sequence of events may require one set of tasks to execute, while another sequence may require another set of tasks to execute. Thus, a provision has been made in ACES to allow the user to perform a simple overlay structure thereby conserving memory requirements. The overlay structure must be predefined at linkage edit time. Each group of tasks which constitutes one overlay segment is called a Job Phase. A Job Phase is composed of one or more tasks and is resident on bulk storage until needed. Figure 3-1 is a diagram of the overall concept of Jobs, Job Phases, and Tasks.

The main segment is loaded when a job is initiated. A task in the main segment must be predesignated to be scheduled immediately by ACES upon job initiation. It is this initial task's responsibility to begin the job's task scheduling mechanisms and to request the initial Job Phase to be loaded.

TABLE 3-1 ACES REQUEST SERVICES

Job Schedule	Open File
Job Terminate	Close File
Job Cancel	Buffered I/O
Job Phase Load	Direct I/O
Task Schedule	Get Main Memory
Task Terminate	Free Main Memory
Abnormal Termination (ABEND)	Boundary Mover
Task Cancel	Lock Variable
Task Status	Unlock Variable
Wait Call	System Subroutine Call
Alert Call	System Subroutine Complete
Event Set	Time Request

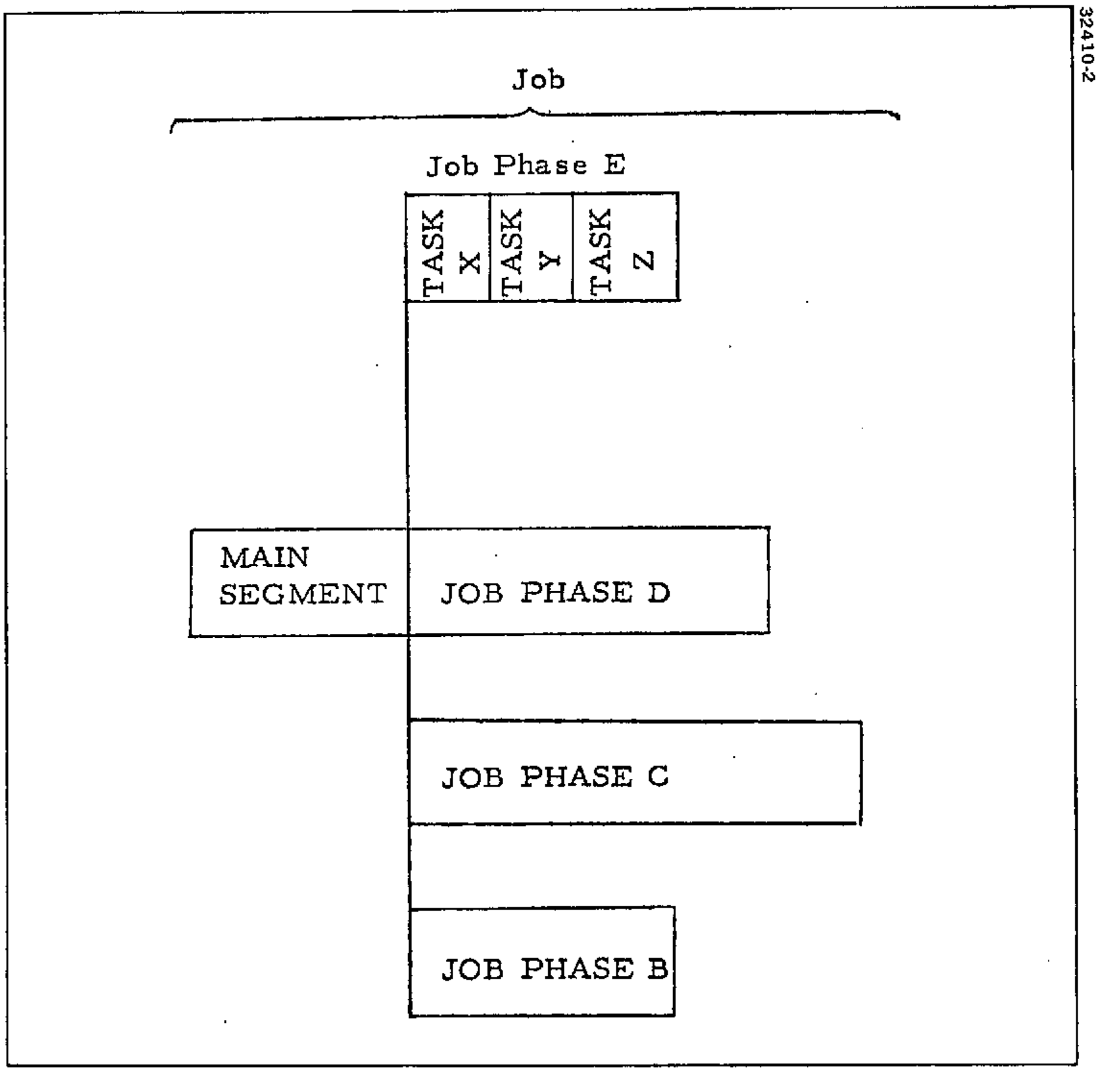


Figure 3-1. Typical Job Layout

Furthermore, it is the responsibility of this initial task to control all other phases loading requests. A Phase Load request may only be executed when all tasks within the currently loaded phase have become quiescent.

The following are the user's Job Control request services supported by ACES.

Job Schedule

The Job Schedule request allow the user to schedule a job for execution. The job requested for execution is placed immediately into a job queue. Jobs are selected out of this queue based upon priority and resources.

Job Terminate

A Job Terminate request specifies that a job is to be terminated. Any task within a job can request termination of that job. A task of one job cannot terminate another job.

A Job Terminate request causes all of a job's tasks which are scheduled, to be deleted. The tasks which are currently executing or in the wait state are allowed to proceed until they terminate. No re-scheduling of periodic tasks is performed after the Job Terminate request is received. When the last task of a job terminates, the job is removed from the system and the memory partition made available.

Job Cancel

The Job Cancel request allows a task of a job to cancel a previously scheduled job. If the job specified is in the job queue, it is deleted. However, if the job is not found in the queue (previously executed) or if it is currently active, the request is ignored.

Job Phase Load

The Job Phase Load request allows a task in a job to request the loading operation of another job phase of the same job into main memory. The request may only be performed when all tasks of the currently resident job phase are quiescent.

3.3.2 Task Control

The ACES operating system is intended primarily to provide a reliable environment for real-time jobs. Since such jobs are generally composed of many independent tasks, considerable effort has been expended to provide a powerful, convenient system for managing such tasks. The system provides a scheduling facility which, coupled with ACES' unique dispatcher, allows the application designer to make effective use of the redundant and parallel processing capabilities of ARMMS hardware. To control these facilities, ACES responds to several requests.

Task Schedule

In order to enter execution, a task must be scheduled. There are several ways in which a task may be scheduled:

1. Immediately.
2. After the occurrence of specified event(s).
3. After a specified time.
4. Combination of 2 and 3.

If neither a time nor a list of events is specified in a Task Schedule request, the scheduled task enters contention for execution immediately. Once it enters contention for execution, it is chosen for dispatching based on its priority and the availability of sufficient resources (CPE's).

A Task Schedule request may specify a list of events and a minimum number of events. In this case the task does not enter contention for execution until after the minimum number of the listed events has occurred.

A Task Schedule request may also specify a time to execute. In this case, the task enters contention for execution at the specified time or after the specified time interval.

If both a list of events and a time to execute are specified, the system first processes the time requirement before beginning to monitor for the specified events. If the application designer wishes to monitor for the events during the time expiration, this can be accomplished by the utilization of an Alert. An Alert request will begin monitoring an event as soon as it is issued. The task schedule may then be based upon the status of the Alert.

In addition to the above request types, if a task has the periodic attribute specified in its Task Dictionary entry, ACES will automatically reschedule the task repetitively. The specified period of the task is the time between one scheduled execution (not entered execution) and the next. If the task has not completed a previous execution when its next period occurs, it is not scheduled for that period. Moreover, if a task does not complete execution for several periods, all executions of those periods will be missed. After completing execution, the task will automatically be rescheduled for the next time period which has not already passed. If a periodic task is scheduled with wait items and/or a specified time, these apply only to the first execution. After the first execution the periodic scheduling continues until the task is cancelled or the job terminates. If a periodic task ABEND's during one execution, periodic rescheduling will still continue.

Task Terminate

When a task has completed its processing, it concludes with a Terminate request. A Terminate request by a task indicates to ACES that a task has completed execution and that its resources may be freed. If a task is not periodic, it is deleted at this time. If the task is periodic, it is rescheduled for the next future period of execution. Task Termination is an event noted by ACES. Any task may wait for another task's termination.

Abnormal Termination (ABEND)

When a task finds itself to be in error, it may request an ABEND instead of a normal termination. ABENDing of a task is a different event than the normal termination of the same task and may be used to signal special error handling in the application program. A task may also specify an Abnormal Exit Routine (AER) to be performed in the event of an ABEND. The AER allows a task to perform special cleanup operations in the event of an abnormal end.

Several conditions can cause the system to force an ABEND for a task:

- o Irrecoverable hardware error
- o Software error
- o Task timeout

Most hardware errors allow automatic recovery. However, for tasks executing on simplex CPE's or having variable data in simplex modules of main memory, there are some errors which do not allow transparent recovery. In these cases, the application designer must provide recovery procedures.

The CPE hardware detects several types of software errors such as illegal operation code, illegal address, divide by zero, etc. Through the Task Option Table, the user has the option of ignoring these errors, providing his own routines to handle them, or allowing the system to ABEND his task when they are detected.

If a task remains active, either executing or in the wait state for an excessive length of time, the system will force it to ABEND. A timeout check is performed periodically at a rate set by the application designer. Any task which remains active; i.e., has not terminated, for two successive timeout checks will be automatically ABENDed. If it is necessary for a task to wait for an extended period, it should do so through the use of the scheduling facilities, rather than the wait facilities, in order to avoid a timeout ABEND.

Task Cancel

A Cancel request is used to delete a previous Task Schedule request. If the task has already begun execution, the Cancel has no effect unless the task is periodic. If it is periodic, its periodic rescheduling will cease.

A Cancel request may specify either a task name or a task name and number. If only the task name is specified, the Cancel applies to all scheduled requests for the named task. If the task name and number are specified, the Cancel applies only to the scheduled requests referencing that specific task name and number.

Task Status

In order to complete the capability to control tasks, it is necessary to provide the user with a means of ascertaining any task's current status. This is provided by the Task Status request. This request allows the user access to the status flags of the task's control information. These flags indicate whether the task is scheduled, pre-empted, active, waiting, etc. The current status of a task may be important to another executing task.

3.3.3 Event Processing and Recognition

Event Recognition and Response Processing consist of the algorithms and design concepts required to:

- o Allow application tasks to establish a system requirement to monitor and record specific event occurrences.

- o Allow ACES to initiate specific application and system tasks in response to dynamic event occurrences.
- o Allow application tasks to set and/or interrogate the condition of defined events during execution.

An event is defined to be any occurrence for which monitoring logic has been provided in the ACES. Currently defined events are, for example:

- o <u>Task Termination</u> - a specific task has terminated.
- o <u>Task ABEND</u> - a specific task has abnormally ended.
- o <u>Program Flag</u> - a program flag has been either set or reset.

Some events are single shots, while others are flip-flops. For example, the Task Termination event is a single shot. That is, once it occurs, it is irreversible. Thus, the event status cannot, once satisfied, become unsatisfied. Conversely, the Program Flag event is a flip-flop event. One task may set the flag at one point and later another task may reset the flag.

Basically, ACES Event Processing logic provides application tasks with two separate mechanisms, Waits and Alerts, to initiate controlled response activity as the result of an event occurrence. In reality, the two mechanisms are closely interwoven to perform overall event monitoring. However, for ease of understanding, each is discussed below.

<u>Wait Processing</u>

The Wait Call request allows a task to request that ACES place it into a wait state until specific events, specified by the calling task, are completed (occur). A calling task can specify any number of events which must be completed before ACES may reactivate the task. In addition, the calling task may request that a limited number of the specified events cause reactivation. This, for example, allows a task to specify ten events to ACES, but state that when <u>any</u> five of the events are satisfied, the task is to be reactivated.

In addition, the task can wait for a specified period of time. This period of time may be specified as an absolute time or a time interval.

ACES allows a time specification simultaneously with event specifications. In this case, the time expiration will occur before the events are monitored for completion. In other words, while the time period is expiring, the events will be disabled and not monitored for completion. After the time period has expired, the events will be enabled by ACES and monitoring for their completion will begin at that instant. This processing is identical to the manner in which a Task Schedule request handles simultaneous time and event specifications.

Alert Processing

Alerts provide a means of requesting ACES to monitor an event for the user without having the user enter the wait state. An Alert request specifies an event to be monitored and a name to be associated with that event monitoring. Any event may be monitored for the user by ACES. A unique name must be assigned to each Alert so that a user can specify to ACES the exact event monitoring desired. For instance, Task A may request an Alert for monitoring an event early in a mission (name A'); Task B may request an Alert for the same event much later in the mission (name B'). It is possible that the status of A' and B' may be different thereafter since the event could have completed after A' and before B'. The unique Alert names allow the user to specify which Alert is desired, since several Alerts may be monitoring the same event.

At any time after the Alert request, the user can query its status by specifying the Alert name in an Alert Status request. In addition to receiving the status, when complete the user receives a count of the number of times the event has been noted complete since the Alert was initially set up. During critical time phases, not only the event's status but, when complete, the number of times an event has completed may be important to the application.

The user may at any time request ACES to stop monitoring an event by issuing a Cancel Alert request specifying the Alert name.

In addition to the three Alert commands (Initialize, Status, and Cancel) specified above, the Alert is useful in another manner. Whenever a wait is desired for both time and events, the wait processing does not start monitoring for event completions until the time period has expired. In cases where the user desires to have the events monitored during this time period, the following procedures can be performed. First, an Alert request is made for each event to be monitored during the time expiration. Then a Wait request is issued specifying the time and events to be waited for completion. However, the events for which waits have been requested do not directly specify the events to be monitored, but rather contain Alert names as the events. ACES, after the time specification has expired, scans the events to be waited for, to determine if Alert names are specified. If so, the names are looked up in the Alert file, and the status of the wait event is set to the current Alert status. Thus, the Alert allows the monitoring for an event during a time expiration.

Event Processing

ACES receives requests from CPE's and IOP's that it note that events have occurred. These Event Set requests provide ACES a mechanism for knowing when events occur. Whenever an Event Set request is entered into ACES, all Alerts and Wait Events are scanned to update the status of the events being monitored.

When all events specified by a task when entering the wait state have completed, the task's "wait state" is removed and it competes with other tasks, by priority, for dispatching.

3.3.4 Input/Output Processing

The ACES I/O system provides two distinct I/O facilities; first, a simple streamlined access scheme to perform I/O to real-time devices requiring only a few words of data; secondly, a more complex, multibuffering access scheme for devices requiring a transfer of many words of data. Additionally, provisions have been made available for the future addition of a FORTRAN-type format control system and/or a bulk file management system.

Both types of I/O currently supported by ACES perform I/O through files. The following explains the ACES file philosophy.

File Manipulation

All I/O requests in the ACES system must reference a file. Each file which may be used by a job must have an entry in the job's File Description Table. A file description includes its name, current status, pointers to its buffers, logical device number, etc.

A file belongs to a job and may be used by any task within the job. Any task may Open, Close, or access any file belonging to its job.

Before any I/O can be performed on a file, the file must be Opened. This causes buffers to be allocated and initialized, and the logical device to be allocated for use. These resources remain allocated to the file until it is Closed. When it is no longer needed, a file is Closed to release its resources. When a job is terminated, any Open files belonging to it are automatically Closed by ACES.

1. Open File

This request initializes a file for I/O operation. For a buffered I/O file, the buffers are allocated and, for input files, the input buffers are primed.

Also, during the Open operation logical devices are allocated. Logical devices may be allocated for either shared or exclusive use. An Open request may be denied if the logical device is not available. This may occur if:

- o The device has failed and there are no alternatives,
- o The device is requested for shared use and another user has it for exclusive use, or
- o The device is requested for exclusive use and another user has it for either shared or exclusive use.

2. Close File

This request makes a file unavailable for I/O operations through a File Description Table. The logical device and any core buffers used by the file are deallocated and are immediately available for other uses. Any I/O operations outstanding on the file when the Close is requested are cancelled immediately.

Data Manipulation

As stated above, ACES provides for two types of I/O service requests. The following describes these two I/O requests.

1. Buffered I/O

The Buffered I/O request allows the user to access data in buffered I/O files. Through the use of its three options (Release, Get, and Wait), the user may control the operation of I/O.

All I/O buffers belong to the system. At any time the user may obtain possession of one of the multiple buffers belonging to a buffered I/O file. While it is in the user's possession, the user may manipulate the data in the buffer in any manner. When the user is finished with it, a Buffered I/O request with the Release option is performed. This releases the buffer to the system which will proceed to perform I/O on it. For input, it will fill the buffer with new data; for output, it will write the data to the specified device. When the Get option is specified in a Buffered I/O request, the system will examine the next buffer. If it is ready for the user (I/O complete), a pointer to the buffer is returned; if the buffer is not ready, the Wait option is examined. If the Wait option is not set, the routine returns to the user with an indication that the buffer was not available. If the Wait option is set, a Wait Call request is performed for the user causing the task to enter the wait state awaiting I/O completion on the pending buffer. When the buffer is ready and the task reactivated, control is returned to the caller with a pointer to the buffer.

2. <u>Direct I/O</u>

The Direct I/O request allows an efficient means to perform I/O where only a few words of data are to be transferred. At any one time, the ARMMS system may accommodate only one Direct I/O request. If additional requests are made by other processors, they will cycle awaiting availability of the Direct I/O facility. This facility bypasses the normal buffering and queuing mechanisms of the ACES I/O system to allow the user to read or write a limited amount of data to/from a real-time I/O device. The user must provide any and all buffer space needed.

3.3.5 Resource Control and Services

ACES controls the various resources needed to execute application programs and provides a variety of user utility services. This section describes the control of several resources and services not described elsewhere in this document.

Main Memory Resource Control

ARMMS main memory is divided into two categories - ACES memory and user memory. ACES memory occupies contiguous address space and is always resident in the maximum criticality logical memory allowed by ARMMS. User memory comprises the rest of available logical address space and is subdivided into four partitions each of which may accommodate one job. Individual modules (8K words) within a partition may, hardwarewise, operate in a simplex or duplex mode. After a job is loaded into a partition, the rest of the partition is available to the user as dynamically allocated memory. Several services are provided to the user to control memory allocation.

1. <u>Get Main Memory</u>

This request service allocates an area of dynamically allocatable memory to a task for temporary storage. A task may have, at most, one such temporary storage area at any one time. This is primarily due to the hardware constraint of one temporary storage base/bound register. This facility is also used to provide a temporary area for I/O buffers.

2. <u>Free Main Memory</u>

This service allows the user to inform ACES that the temporary storage area previously allocated to the requesting task is no longer needed and may be released. The temporary storage base/bound register is reset.

3. Boundary Mover

The boundaries of the four partitions are initially set at system start-up time. Thereafter, the user may change the boundaries at any time. The Boundary Mover request allows the user to move the boundary between two adjacent partitions. A boundary can be moved only into an empty partition; i.e., if the partition is to be moved to a lower address, the lower partition must be empty. One of the criteria for loading a job is the availability of a partition of sufficient size. The Boundary Moving request is provided so a user can dynamically control the partition size, therefore, increasing job throughput by knowing the system requirements during a given time period.

Information Protection

A system of interrelated tasks must have shared data. This sharing of data creates a potential for access conflicts among cooperating tasks. The ACES system provides a means of control for such conflicts through the Locked Variable request service. Any contiguous set of shared data locations may be "Read-Locked" or "Write-Locked".

A read-lock, applied to a set of data, prevents any other task from modifying that data set until the read-lock has been removed. A write-lock, applied to a set of data, prevents any other task from reading that data set until the write-lock has been removed.

To accomplish the locking, ACES uses "Lock-Variables". A lock-variable is a memory location that contains lock information pertaining to a contiguous set of shared data locations. To facilitate their use, a hierarchy of lock-variables may be defined as depicted in Figure 3-2. Two services are provided to control the data lock facility.

1. Lock Variable

This request applies a lock to a Lock Variable. If the lock cannot be granted immediately (due to the variable being previously locked), the task will be notified. It is the user's responsibility to enter the wait state, awaiting an unlock of the variable, if no further execution can be performed until the lock is obtained.

2. Unlock Variable

This request removes a lock placed on a data lock by a Lock Variable request. Any tasks awaiting the variable to become unlocked will be removed

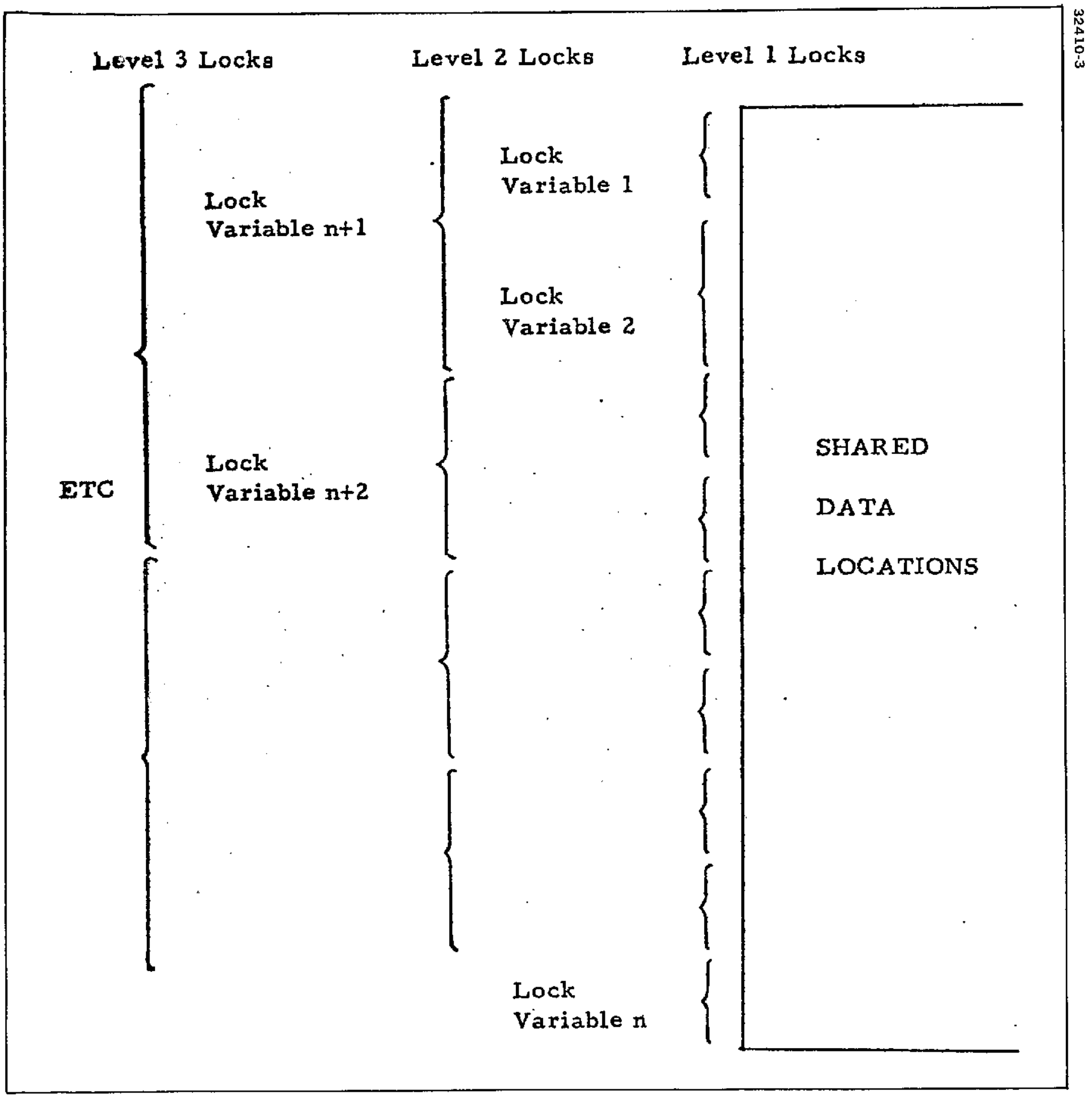


Figure 3-2. Lock Variable Usage

from the wait state.

System Subroutines

ACES provides a service for managing the sharing of common subroutines among independent tasks. (This service is not for calling a task's own subroutines.) Two services are provided to control the system subroutine facility.

1. System Subroutine Call

Certain System Subroutines which are common across many jobs are included in ACES domain. Other System Subroutines may be included in individual jobs. When the jobs are loaded, the System Subroutine list is provided to ACES. Then a task issues a System Subroutine request to ACES when a subroutine's execution is desired. If the subroutine is unavailable (non-reentrant and in use), the task is notified that the subroutine is not available. It is the user's responsibility to request a Wait Call if no further execution can be performed until the subroutine is available.

When the System Subroutine is available, all of the task's current environment (registers, program counters, etc.) is saved by ACES. The System Subroutine is initiated by ACES with the subroutine's own base/bound registers, program counters, etc. The only item transferred between routines is the address of the parameter list (if one). The System Subroutine is executed on the same stream that requests the service to provide efficient response time and have the routine execute at the same criticality as the originating task.

2. System Subroutine Complete

Each System Subroutine must issue this request at its termination. This request signals ACES that the routine is complete and is available for another request. ACES then reloads the processor(s) with the task's original environment and restarts the stream.

Time

ACES maintains a real-time clock which is used in many of its scheduling functions. This clock is made available to the application programs via an ACES service request. The basic resolution of this clock is 100 µs. The format in which the time is returned to the user is variable and dependent on the requirements of the application.

3.3.6 Fault Processing

Fault Processing is an integral part of the ARMMS project. ACES has been designed with fault processing as one of the major items to be considered in every program's design. The following summarizes the ACES fault processing philosophy.

Fault Processing Overview

ACES fault processing depends heavily on the excellent fault detection facilities of the ARMMS hardware. Virtually all hardware faults are detected by the hardware, which notifies the BOSS processor via an interrupt. Within the BOSS processor, ACES software analyzes the faults and takes appropriate diagnostic actions.

ACES first attempts to recover the operation of the affected task. In most cases, register data can be recovered and saved for the dispatching system just as if the task's execution had been pre-empted by a higher priority task.

Next, all faultless hardware must be placed back into production. Any module(s) which are not suspect can be immediately returned to production. For instance, if a duplex processing stream detects a discrepancy and halts, and one of the processors can immediately be identified as at fault (from hardware indications), the other processor can be returned to production immediately. Diagnostics are performed to verify the existence of a failure. If the failure cannot be reproduced, it is assumed to have been transient and the module is then returned to production.

When a module fails, an attempt is made to replace it from the spare pool, powering up spare modules if necessary.

If a module has indeed failed, and there is no spare to replace it, then the capabilities of the system are reduced and steps must be taken to reduce the CPE work load. This is accomplished by calling the Task Dictionary Decrementor to cause a job to step to a Task Dictionary of Lower Levels (DOLLs). Each dictionary specifies the tasks that are valid during its dictionary period and the minimum hardware requirements (resources) necessary for the DOLL to be meaningful. Each succeeding DOLL requires less resources than its predecessor. DOLLs provide a job a means of decreasing its processor work load by specifying fewer tasks, etc., as resources decrease rather than immediately aborting or decreasing its efficiency to the extent that deadlines cannot be met.

When a job is initially loaded, its DOLLs are examined, and the highest level dictionary which the available hardware will support is initiated. Thereafter, whenever a hardware resource fails, the Task Dictionary Decrementor is called, the task dictionaries of all active jobs are examined, and the highest level which the currently available hardware will support is chosen for each job. If any of the dictionary requirements for a job cannot be met with the available hardware, the job will be deleted.

When it is not busy with detected faults, the fault processing system performs periodic diagnostics on all modules - active, spare or failed. It is possible for such testing to locate a failed module which has become functional again. When this happens and the module is put back into production, the Task Dictionary Incrementor is called to adjust all job's DOLLs to their highest possible level to make full utilization of the newly available hardware.

<u>System Initialization/Restart</u>

The ACES initialization/restart facility is provided to initially start the ACES system or to restore the system after a massive failure or transient which has caused ACES to function improperly or stop functioning entirely. The following briefly describes the means by which ACES is initialized or restarted.

First, the system is cleared of any active jobs, tasks, wait items, and Alerts. Next, all hardware modules except BOSS and ACES main memory are cleared and all ACES tables and queues are initialized or re-initialized. An operable set of CPE's, memories, and IOP's is then located (by performing diagnostics) and configured. Once the system itself has been restarted, it is possible to begin execution of jobs from the Job Queue.

3.4 ACES Program Description

This section presents the detailed functional design of the ARMMS Control Executive System. The material in this section is intended to provide a summary of the overall executive program logic.

Each of the following subsections discusses the major software functions to be performed. Where necessary for clarification, individual routines are discussed. Individual program module descriptions and flow-charts are not included in this document. They can be found, however, in M&S Computing Report No. 73-0018, "Complete Executive Detail Design Final Report", prepared for the Hughes Aircraft Company.

Before proceeding into individual subsections of ACES a few comments are applicable to the entire system.

- Priority Structure

  The priority structure for ACES is most simple in nature. There are four levels of priority. The following describes the purpose of each priority level from the highest to lowest level.

  - Initialize Reset - This priority level is the highest available. This level, when activated, will cause any lower level priority levels to be suspended. The purpose of the level is to perform BOSS Initialization or Reset. Upon the initial power up sequence, ACES must perform several basic house-cleaning functions. These functions are the same as those needed if massive hardware and/or software failures occur which exceed ARMMS failure correcting capability. Upon receiving control, this priority level resets all ACES tables and begins to establish control of the entire system.

  - Timer Level - This priority level is responsible for updating all software clocks from hardware timer interrupts. The level is a high priority level since an extremely fast response time is critical to maintaining an accurate time over a five year mission.

  - Request Level - This level processes all fault detection and service requests from CPE's and IOP's. Approximately ninety percent (90%) of all ACES software modules execute at this level, therefore, it is the major level with which ACES is concerned. Any fault detection mechanism or service request will cause this priority level to become active. By handling these functions at this level, an efficient response can be provided to both.

  - Diagnostic Level - This fourth and lowest level is not activated due to an interrupt. It is the base or background level for ACES. The level is continuously looping, looking for faults which may have previously been undetected and performing diagnostics when no other ACES function is needed. The level is only active when no other priority level is processing.

- Layering

  In designing ACES, considerable emphasis has been placed on software reliability. Layering is a new concept within the programming environment whose goals (simplification of maintenance and verification, and increased system reliability) are synonomous with ACES.goals. It therefore is highly desirable to attempt to incorporate this technique into the detailed functional design development effort.

The layer concept attempts to force certain structuring upon the software design. This software structuring forms layers of "levels of abstraction". Each layer includes one or more related software components which share common data. Logically, layers are stacked upon each other to form a hierarchical structure. Each layer in the hierarchy performs a unique function and has its own exclusive resources. The lower the layer is in the hierarchy, the more closely associated with the actual hardware are its components. Figure 3-3 shows a common layering example in which components in the top layer perform content addressing while the lowest layer performs physical addressing.

Figure 3-4 presents a pictorial view of some of the basic groundrules of layering. First, components within one layer may reference components only in lower layers, not in higher layers. Secondly, a component in one layer may directly reference its own layer's resources (devices, data, etc.), but not resources of another layer. However, if a component in one layer needs information (data) available in a lower layer, it may call a component in the lower layer and request information available there. Components have knowledge only of components in lower layers; never can a higher layer resource be obtained.

One of the advantages of layering is the ease of checkout. Layers are checked out beginning at the lowest layer. Once that layer has successfully been tested, the next highest layer may be added and tested. Since each layer is logically independent of upper layers, software "bugs" should only be found in the newest layer to be tested.

Figure 3-5 presents an overall view of the ACES layering scheme. Table 3-2 details individual routines within each layer. Considerable effort has been expended to insure its correctness and validity. In addition to the partitioning of all ACES modules into layers, it should be noted that a functional separation of fault detection/recovery from the executive services has been performed. This was done to insure that these services could easily be divided into separate hardware modules if future ARMMS requirements dictate. It should also be pointed out that the Interrupt Management layer (layer 0) is logically separated from the other layers. This was performed to insure its independent operation from both executive services and fault detection/recovery. If these services do become divorced from a single

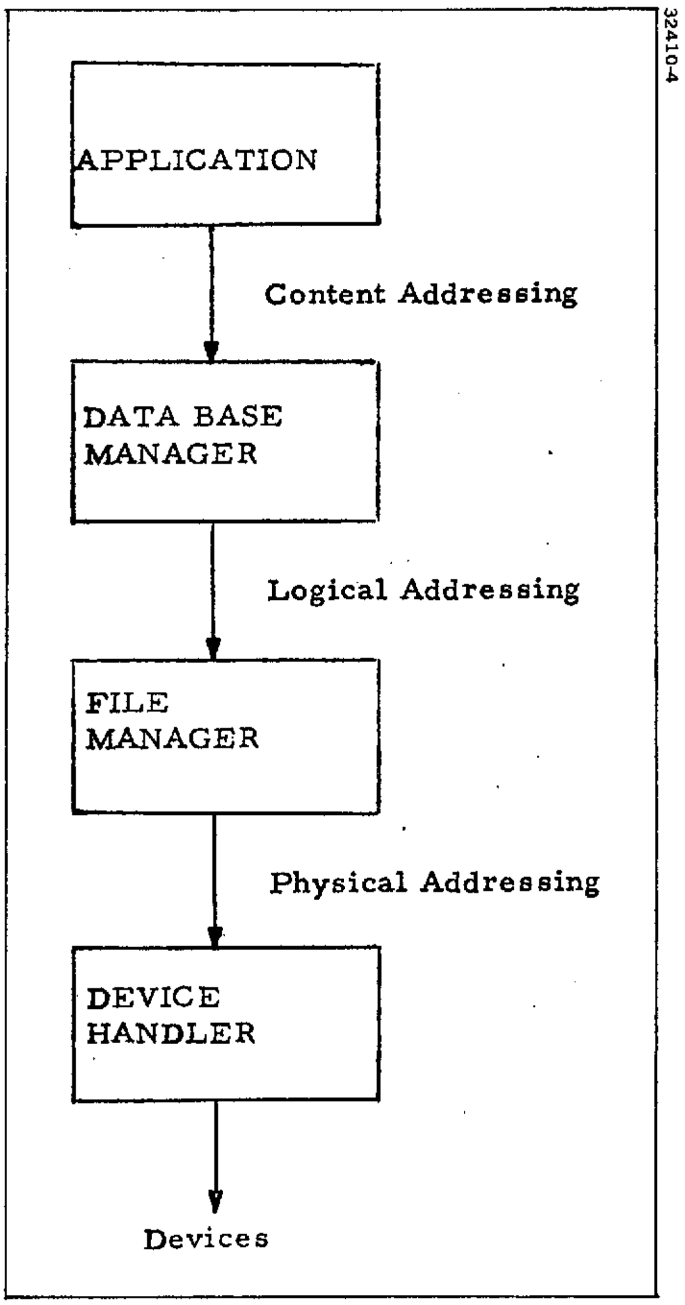


Figure 3-3. Common Layering Example

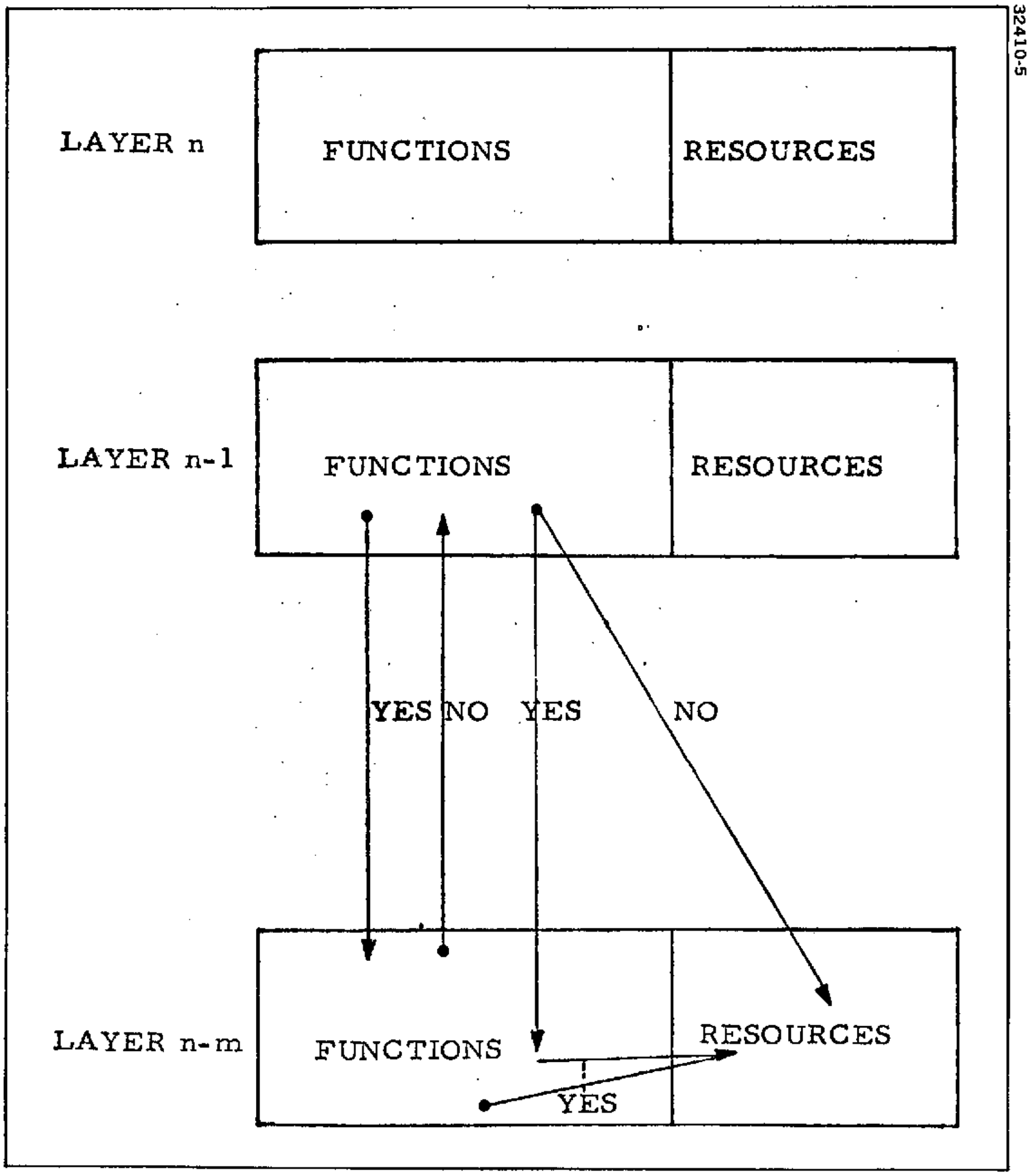


Figure 3-4. Layer Groundrules

ACES LAYERING STRUCTURE

EXECUTIVE SERVICES		FAULT DETECTION/RECOVERY	
LAYER	RESOURCES	LAYER	RESOURCES
10. Request Management			
9. Job Management	JPQ JAL JIB JDF		
8. I/O Management	I/O Request Queue I/O Priority Queue		
7. Service Management	Lock Variable Table Subroutine Table		
6. Time Management	Software Clocks		
5. Scheduling Management			
4. Event Management	File Memory		
3. Task Resource Management	LAAT Module Status Table		
2. TD-TQM Management	Task Dictionary TQM		
1. (A) Initiation Management	Master Execution Table	(B) Diagnostic Management	Module Status Table Master Execution Table

0. Interrupt Management	R. T. Clocks Interrupts

Figure 3-5

ACES LAYERING COMPONENTS

LAYER	PROGRAMS	MAJOR TABLES
10. Request Management	Request Processor	
	Special Request	
	Diagnostic Request Processor	
9. Job Management	Job Scheduler	Job Information Block
	Job Activator	Job Priority Queue
	Job Initiator	Job Dictionary File Index
	JPQ Searcher	Job Active List
	Job Active List Maintenance	Job Dictionary File
	Job Terminator	
	Job Cancel	
	Task Dictionary Increment	
	Task Dictionary Decrement	
	Change Task Dictionary	
	Search Task Dictionary	
	Job End	
	Timeout	
	Task Terminate	
	Abnormal End	
	Abnormal End Initiate	
	Job Terminate Cleanup	
	Task Terminate Cleanup	
	Job Phase Loader	
8. I/O Management	File Open	Channel Status
	File Close	I/O Request Queue
	Close all User Files	I/O Priority Queue
	Buffer Control	Physical I/O Device
	Buffered I/O Request	Buffer Description
	Device Control	
	Direct I/O Request	
	IOP Main Cycle	
	DIO Checker	
	Queue Mover	
	Channel Initiator	
	I/O Finish	
	Normal I/O Finish	
	Retry Processor	
	Select Alternate Device	
	I/O Error Logger	
	Cancel I/O	

Table 3-2

ACES LAYERING COMPONENTS
(continued)

LAYER	PROGRAMS	MAJOR TABLES
7. Service Management	System Subroutine Call System Subroutine Complete Lock Variable Unlock Variable Get Main Memory Free Main Memory Memory Partition Allocation Partition Deallocation Partition Boundary Mover Job Resource Comparator	Subroutine Call List Lock Variable Table Memory Partition Table
6. Time Management	Timer Processor Timer Queue Processor Clock	Software Clocks
5. Scheduling Management	Task Scheduler Timer Scheduler Priority Scheduler Find TQM Slot Return TQM Slot Wait Call Processor	
4. Event Management	Wait Event Processor Alert Event Processor Alert Call Processor Alert Terminate Alert File Scan Enter Wait Items Wait File Processor Turn on Wait Items Disable Wait Items Find File Memory File Memory Maintenance Delete Wait Items Return File Memory	File Memory

Table 3-2
(continued)

ACES LAYERING COMPONENTS
(continued)

LAYER	PROGRAMS	MAJOR TABLES
3. Task Resource Management	Task Cancel Task Status Task Dictionary Comparator Job-Task Halt	LAAT Unit Status Table Module Status Table
2. TD-TQM Manager	Task Dictionary Manager TD Entry Read TD Entry Write TQM Manager TQM Read TQM Write Link/Delink Priority Queue Link/Delink Timer Queue TQM Maintenance Pre-dispatcher Dispatcher	Task Dictionary Task Queue Memory
1-A Initiation Management	Start Task Configurator Table Update Stop Task Reservation Checker Minimum Priority Stream Identification	Available Resource Word Master Execution Table Connect Word Table
1-B Diagnostic Management	Failure Pre-processor Fault Processor Tester Reservation Call Reservation Return Schedule Service Request Memory Failure Processor Page Fault Processor Pager	Master Execution Table Test Information Table Module Status Table Resource Request Word

Table 3-2
(continued)

ACES LAYERING COMPONENTS
(continued)

LAYER	PROGRAMS	MAJOR TABLES
0. Interrupt Management	Interrupt Processor Read MSW Mission Timer Processor Timer Control Start Stream Stop Stream BOSS I/O	Real Time Clocks Interval Timer Interrupts

Table 3-2
(continued)

processor, it is possible that a new layer 0 would have to be designed for each.

The following subsections follow the ACES layering scheme for presentation. It is felt that this manner of presentation is the most valid from the system design point of view and the most meaningful from a reader's viewpoint.

3.4.1 Request Management

The highest layer of ACES is involved in the distribution of service requests to other parts of the system. Requests may come from three sources:

1. Application users.
2. ACES routines at different priority levels (special request).
3. ACES diagnostics system.

All three sources result in the calling of the appropriate service routine to process the request. Since the three sources generate request via differing tables and queues, there are three routines to handle the request. Figure 3-6 depicts a conceptual view of request processing.

The Request Management layer is responsible for calling Dispatcher. Before Dispatcher is called however, the layer insures that all outstanding service requests have been performed. Also, the Predispatching routine must indicate that Dispatcher execution is needed. If not, the Dispatcher is not called.

The following discusses each of the sources that request services via the Request Management routines.

Application User Requests

When an application user (or an IOP) requests a service (e.g., Event Set) of ACES, the Module Status Word (MSW) of the executing processor is modified by the processor's hardware/firmware to contain the request. This changing of a processor MSW causes an interrupt to be generated in the BOSS processor. This interrupt is received by the ACES interrupt handler and the service request is passed to the Request Management routines. The nature of the services needed is specified in the requesting module's MSW. The MSW is divided into two sections; a fault section and a request section. Hardware fault masking makes it possible for both sections of the MSW to

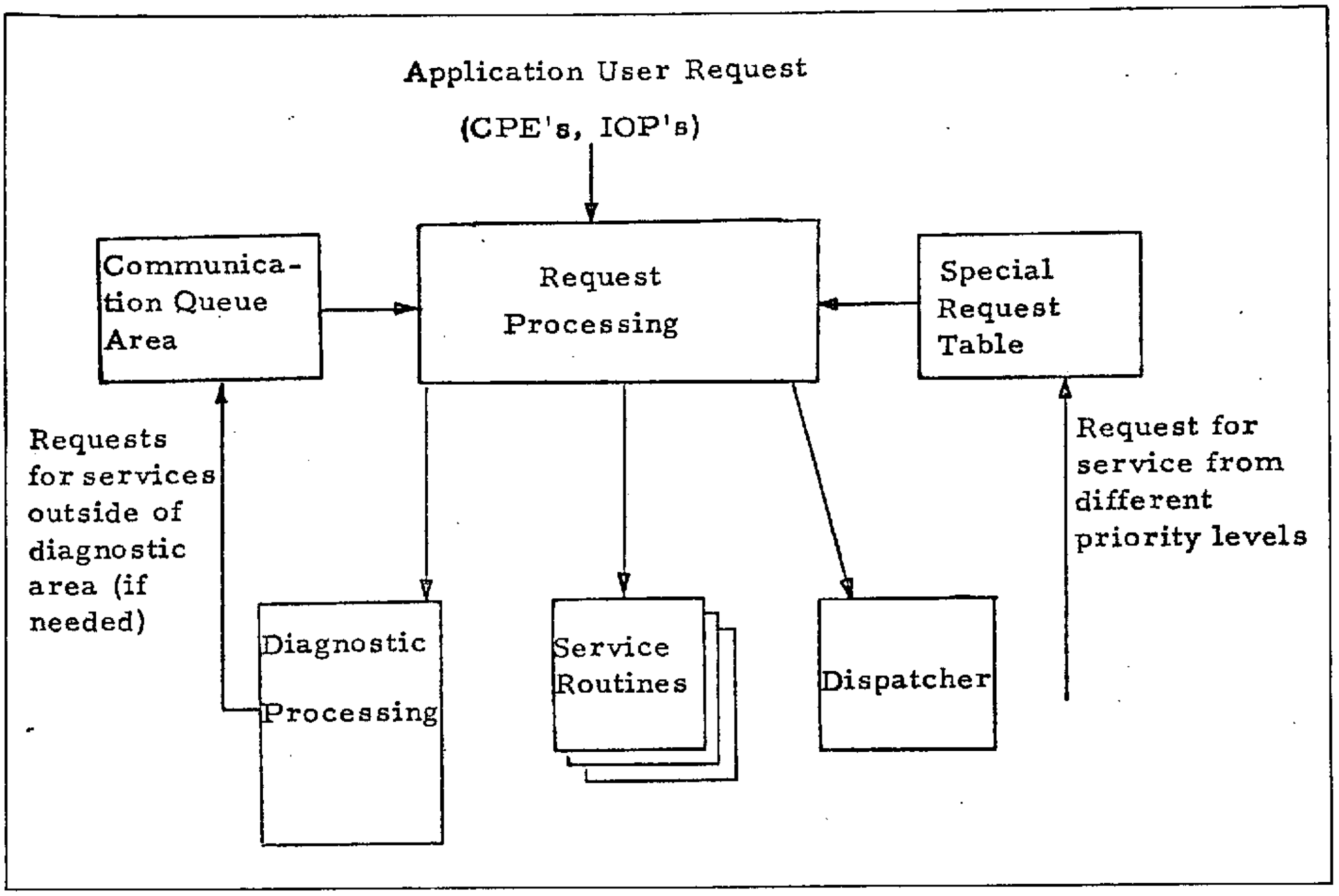

32410-6
Application User Request
(CPE's, IOP's)
Communica-tion Queue Area
Request Processing
Special Request Table
Requests for services outside of diagnostic area (if needed)
Request for service from different priority levels
Diagnostic Processing
Service Routines
Dispatcher

Figure 3-6. ACES Request Processing

contain valid data. The fault section is examined before the request section is examined. If fault processing discovers that the task is irreparably damaged, it can cause the task to be aborted by changing the request code to an ABEND, which will then be processed.

The user Request Management routines insure that all CPE's or IOP's of a stream make the same request and, therefore, are in lock step. If the same request is not made, a fault is noted by the system and the request continues if the proper service request can be determined. The user Request Management routines must, if the request was made by a CPE, determine from which task the request was made. The information along with the task's job ID must be appended to the request so that service routines which process the request can perform validity checking, etc. The job ID must be carried internally by ACES as up to four jobs may be in execution at a time and the user is unaware of the other jobs. Thus, two tasks in separate jobs might request a Task Schedule specifying the same task name, each attempting to schedule a unique task in its own job. It is the system's responsibility to append the job ID to the user's request so that it can distinguish the two separate job requests.

ACES Special Requests

Various parts of the ACES software operate on four different interrupt priority levels. It is sometimes necessary for routines at different priority levels to utilize some of the service request facilities. To avoid the possibility of recursive entries in such service routines, the capability exists whereby an entry is made in the Special Request Table. The Request Management routines, executing at the proper priority level, call the proper routines. This routine performs as a basic scheduling system within ACES.

ACES Diagnostic Requests

The Diagnostic Processing System has been designed to operate as independently as possible from the rest of ACES. It was designed such that it does not need to directly call any ACES routines outside of the Diagnostic Processing section. To implement this scheme it was necessary to provide a means by which the Diagnostic Processor could request execution of user services similar to those described above. This is performed by placing the request for a service into a Communication Queue Area. This service request is acted upon by the Request Management routines the next time the routines are placed into execution.

3.4.2 Job Management

Job - Phase

The structure of a job is defined such that a job consists of one or more job-phases. Each job-phase may consist of one or more tasks.

A job may be defined as having separate and distinct parts, with each part executed in a prescribed sequence. These parts are defined as job-phases, where a job-phase may consist of one or more tasks. Tasks of one job-phase may communicate with tasks of another job-phase but tasks of one job may not communicate or reference task of another job. A job-phase must be activated by an executing task. That is, a task of one job-phase must activate subsequent job-phases. ACES' job-phase activation consists of resolving all references and scheduling the initial job-phase task.

System Task

ACES provides a comprehensive set of commands to drive a user application program. The multitasking, multijobbing facilities allow the user complete flexibility in the design of the application system.

The ACES Task Management feature allows tasks to be scheduled immediately, based upon a future time, and/or designated event(s). Through these facilities the application can implement an efficient multitasking task structure.

The Job Management facility provides similar features for scheduling jobs, but provides it in a different form so that greater latitude may be achieved. This latitude is provided in an ACES concept called the "System Tasks". The following discusses this concept.

The System Task(s) is one or more application tasks written by the user. Unlike other task codes, the System Task's code and associated control blocks are placed into ACES main memory and remain there, permanently resident, throughout a mission. The System Task's function is to control the overall application structure system design. This is accomplished by monitoring time and events, and scheduling jobs, deleting jobs, etc., based upon these conditions. The System Tasks function as any other task; i.e., they compete with other tasks for facility resources (CPE's, IOP's, etc.), they enter the wait state, schedule other jobs, etc. These System Tasks are grouped together to form a job. The only difference between this job and the other job is that this one resides in ACES memory. Residing in ACES memory does not distinguish this job from any other application job. It is only for convenience that the job's code is placed into ACES memory.

Job Scheduling

Requests for job scheduling will be processed by a system level task. Requests for job scheduling may be entered by an executing task through the Job Schedule Request. Figure 3-7 depicts the Job Processing components at a functional level.

An executing task may specify that any job defined by the Job Definition File (JDF) be scheduled (placed in the Job Priority Queue). Job scheduling will be accomplished by the established ACES interface linkage for system services. A job scheduling request will require parameters to identify the specified job.

The ACES job scheduling routine must determine job identity from the request parameters. When the identity is found, the job priority and Job Information Block (JIB) address are extracted from the Job Definition File Index (JDFI). Using the job priority, the JIB address is positioned in the proper Job Priority Queue (JPQ) position, maintaining the priority order of all entries in the JPQ. After the entry is made in the JPQ, the job schedule function is complete.

Job Activations

After a job is scheduled, an attempt is made to activate the job. The ACES job activation routine searches the JPQ for the highest priority job contending for initiation and execution. The associated Job Information Block (JIB) is examined to determine the required initial resources. Current resources are then scanned to determine if enough resources are available to support execution of the job. If sufficient resources are available, the required resources are allocated and execution of the job is initiated by initializing the first Task Dictionary and scheduling the job's primary or initial task. If sufficient resources are not available, the next JPQ entry is determined and its resource requirements are examined.

The JPQ search always proceeds in a high to low (or first to last) order. That is, jobs with higher priorities are considered for execution before jobs with lower priorities. If enough resources are not available, lower priority jobs are then considered. So, any job to be executed is examined first by its relative position in the JPQ (priority) and then by the resources which are available as compared to those required by the job. These resources consist of only those necessary to initialize execution and do not include those dynamically allocated by each task of the job.

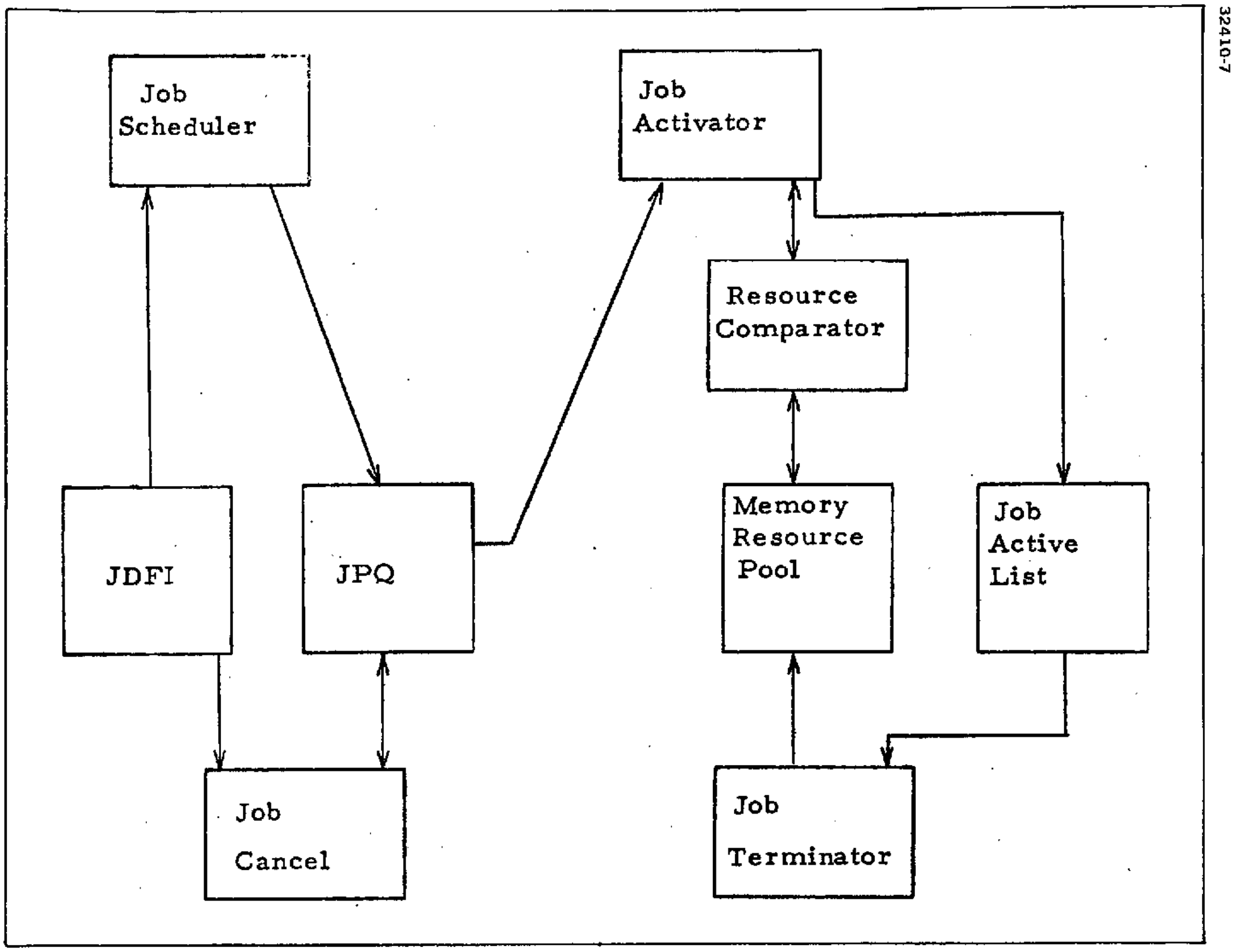


Figure 3-7. Job Processing

This JPQ search process continues until a scheduled job is found that can execute with the resources available. If no eligible job can be found, the job search is terminated and is not started again until either an executing job terminates and frees additional resources or a new request is made to schedule a job.

When a job enters execution, it remains in execution and all resources remain allocated until the job terminates either normally or abnormally. Jobs, unlike tasks, are never pre-empted in order for higher priority jobs to obtain their resources. If jobs having higher priorities are scheduled while lower priority jobs are executing, the higher priority jobs must wait until a job(s) completes execution, if there are not enough resources to support their execution. There is no deviation in the sequence for executing a job. It always is in the following sequence: first schedule, then execute, and finally terminate.

Job Termination

Jobs are never suspended for any reason. When a job enters execution, it remains in execution until it terminates normally or abnormally. As in scheduling, jobs are terminated by an executing task. The task which terminates a job may optionally schedule another job, but it must, in any case, signal the system that the job is normally or abnormally terminating. A task may terminate only the job of which it is a part. Tasks of one job may not terminate other jobs. This is not allowed since errors in one job should not be allowed to propagate to the entire system.

When a job terminates either normally or abnormally, all resources allocated to that job are returned to the system, or de-allocated.

Job Tables

Figure 3-8 depicts the job scheduling intra-table communication. Each job of the system will be defined by a central information file which is called the Job Definition File (JDF). All jobs which are eligible for scheduling are identified and defined by the JDF. The JDF will be built by an off-line system generation function so that during real-time operation every job eligible for execution is predetermined. When the system is operable, JDF is fixed so that job definitions may not be dynamically generated or modified. The JDF will consist of a number of Job Information Blocks (JIB's), each of which will define one complete job. As many JIB's as necessary will be provided for the predicted system application.

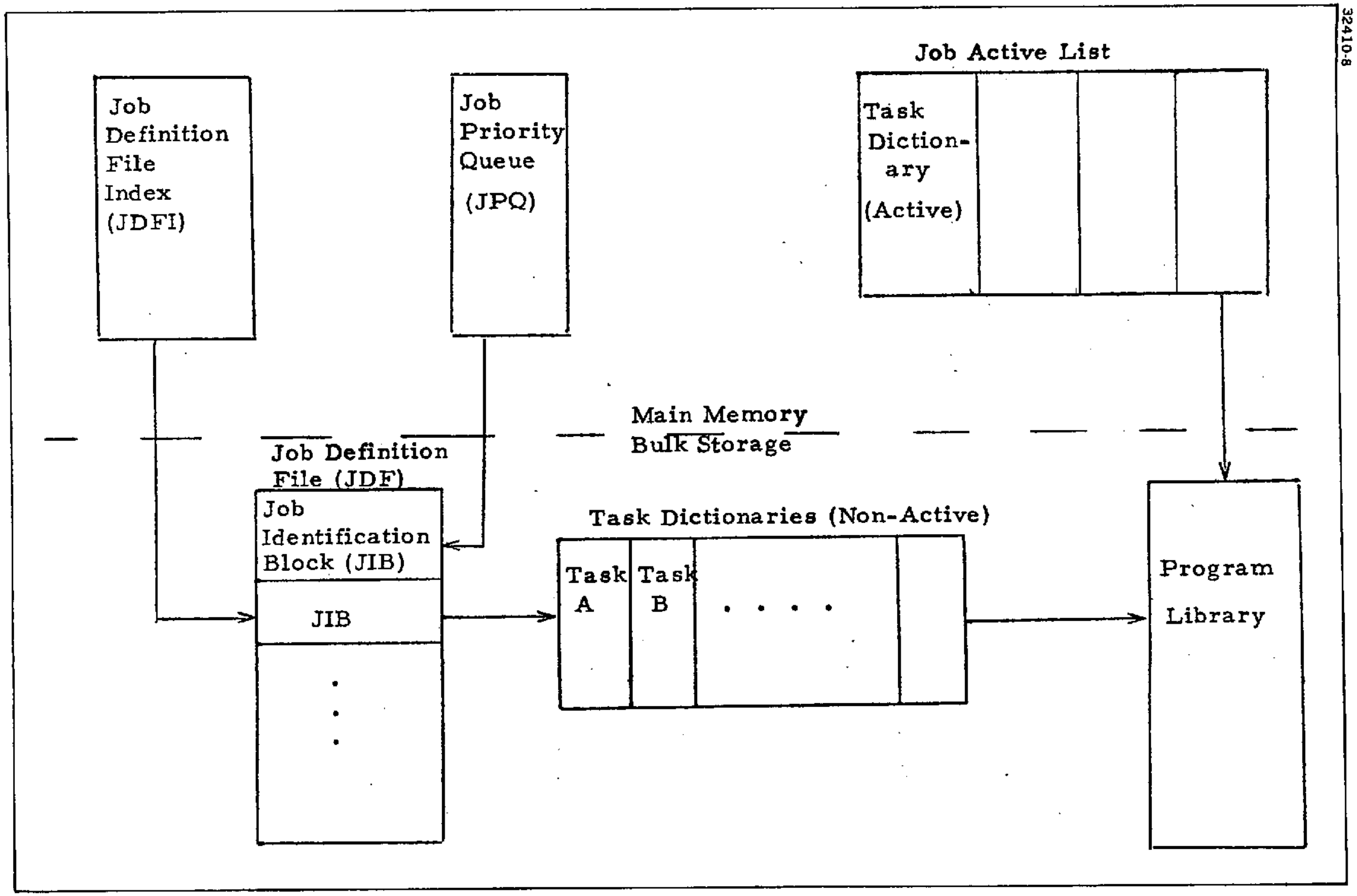


Figure 3-8. Job Processing Tables

To provide an efficient access method for job scheduling functions, a Job Definition File Index (JDFI) will be provided by the system. The JDFI is main memory resident and contains pointers to the JDF, which is normally, due to its size, resident on bulk storage device. The JDFI allows ACES job scheduling to perform efficient validity checks on job names and provides an efficient mechanism for referencing a particular JDF entry.

In order for a job to be eligible for execution, it must first be scheduled. A Job Pending Queue (JPQ) is maintained to provide the system with a current list of jobs that have been requested for execution. Scheduling functions will provide capability to place a request for a particular job execution in the JPQ. The JPQ is an ordered table of pointers to JIB's of each scheduled job. Scheduling a job consists of finding a JDFI entry for a job, picking up the JIB pointer from the JDFI, and entering the pointer in the JPQ in the appropriate priority position.

The JPQ is an ordered list of scheduled jobs such that the highest priority job contending for execution will be the first entry in the queue. Lower priority jobs appear in the JPQ in descending order. Jobs having the same priority are entered on a first-in, first-out (FIFO) basis. Each JPQ entry contains a single parameter which is the address of the JIB for the requested job. The JPQ contains sixteen entry locations. This implies that the maximum number of jobs scheduled at any time is sixteen.

3.4.3 Input/Output Management

I/O Hardware Functional Overview

Due to schedule limitations, the hardware I/O section of ARMMS was not defined at the time the software was designed. Many assumptions concerning the hardware were made and discussed with ARMMS hardware personnel. It was agreed that all assumptions were reasonable and software design should proceed using them. The following briefly describes major hardware assumptions made to design the software I/O system. Figure 3-9 depicts the conceptual ARMMS I/O configuration.

The I/O Processing (IOP) unit is an integral part of the ARMMS I/O system. Each IOP is capable of controlling the Bus Control Unit (BCU). An IOP is expected to be a small computer, a sub-set of a CPE. Unlike the CPE's which are constantly being reconfigured into TMR, duplex, and simplex logical modules for differing redundancy requirements, the IOP's are never reconfigured for different I/O requests. IOP's are configured to function as one logical unit. This logical unit may be composed of one, two, or three (as mission requirements dictate) IOP's functioning as one IOP; e.g., in lock step. The only reconfiguration during a mission is when one IOP of a logical unit fails and is replaced by a spare IOP.

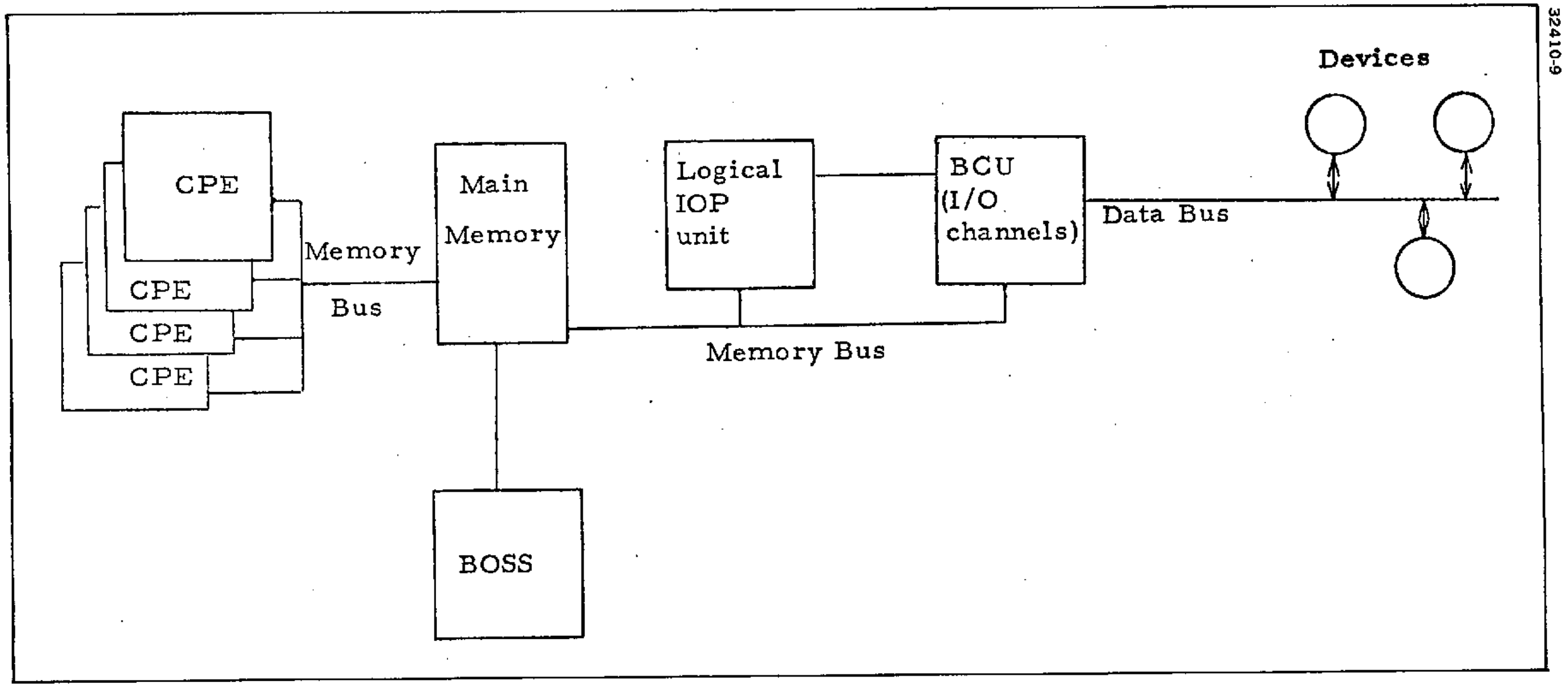


Figure 3-9. Conceptual ARMMS Configuration for I/O

The CPE's and BOSS request I/O services via main memory queues and tables. Here, the I/O operations reference logical I/O devices. Software in the IOP translates logical to physical device numbers, starts I/O operations, handles I/O completions, and retries in case of failures.

The IOP interfaces with a BCU which contains several independent channels. Each of these channels is capable of operating independently of the IOP or other channels to transfer a block of data between memory and an external device via the data bus. All of these channels are functionally identical; any of them may access any area of main memory and any device on the bus. All devices are on the data bus. Any combination of channels may be in operation simultaneously.

A channel begins an I/O operation when it receives an initiate command from the IOP specifying an I/O device, I/O bus, I/O operation code, word count, and starting address. The channel establishes communication with the requested device via the specified bus, transmits the operation code to the device, and when the device is ready for it, proceeds to transfer data to or from the device beginning at the starting address. The channel signals the IOP when the operation is finished due to satisfaction of word count, termination requested from device, or an error which does not allow the operation to continue. At any time, the IOP can perform an inquiry of the BCU to ascertain device address, bus address, error code, and remaining word count for any channel.

I/O Management Processing

The ACES I/O system comprises a group of interrelated software modules executing in the various processors of the ARMMS system. The user, whose task executes in the CPE, interfaces with the I/O system via a group of re-entrant service routines which execute in the CPE. Whenever I/O is desired by the user, the user's executing stream branches to a service routine in main memory. BOSS intervention is not required. These CPE I/O routines receive user request, handle buffers, and request IOP services via the DIO Transfer Area and the I/O Request Queue.

In the IOP, Direct I/O (DIO) requests are handled immediately; I/O Request Queue entries are placed in the I/O Priority Queue to be processed in order of priority as resources become available. I/O completions are processed by the IOP which notifies ACES of the event. The ACES Event Processing system is responsible for restarting any tasks waiting for the completion of that event.

Since Opening and Closing of files causes shared resources to be allocated and deallocated, and thus may propagate failures throughout the system, these services are performed in BOSS.

BOSS also has an I/O capability of its own similar to the CPE I/O capability. BOSS I/O capability utilizes the Open and Close routines to initialize I/O files. BOSS utilizes the I/O Request Queue for individual I/O operations. See Figure 3-10 for ACES I/O System.

CPE Routines

The Direct I/O request routine handles the CPE processing of the DIO facility. This routine locks the DIO facility (waiting if necessary until another CPE has unlocked the facility), makes a request for the IOP to perform Direct I/O, and delays until the I/O is completed.

The Buffer Control routine processes Buffered I/O requests from the user or from (when supplied) the File Manager and Format Control routines. It is organized around the Release, Get and Wait options. The Release option causes buffer rotation and the queuing of an I/O request for the buffer being released. The Get option causes the next buffer to be examined. If it is not ready and the Wait option is set, the Buffer Control routine issues a Wait Call request to BOSS requesting a wait for I/O completion on the next buffer. When this I/O is complete, the task will resume processing in the Buffer Control routine which will then return to the caller with the next buffer.

IOP Routines

The IOP Main Cycle is the scheduling routine for the IOP. It tests for conditions requiring IOP services and calls other routines to handle these services. When there are not outstanding requests for services, the IOP's cycle facility is used to render the IOP dormant.

The DIO Checker routine checks for Direct I/O requests and handles them if enough resources are available.

The Queue Mover takes requests from the I/O Request Queue, where they are placed by other modules, and moves them to the I/O Priority Queue which is used solely by the IOP. This routine is also responsible for translating the user's logical device address to a physical device address for use by other IOP routines.

Whenever there is a channel free, the Channel Initiator is called. It searches the I/O Priority Queue for the highest priority request which is not awaiting a busy device. If an outstanding request is found, it initiates the operation on the first available channel.

The I/O Finish routine is called if there are one or more channels with a finished status. This routine determines the status of the operation and calls an appropriate routine to handle the various conditions. If no error is

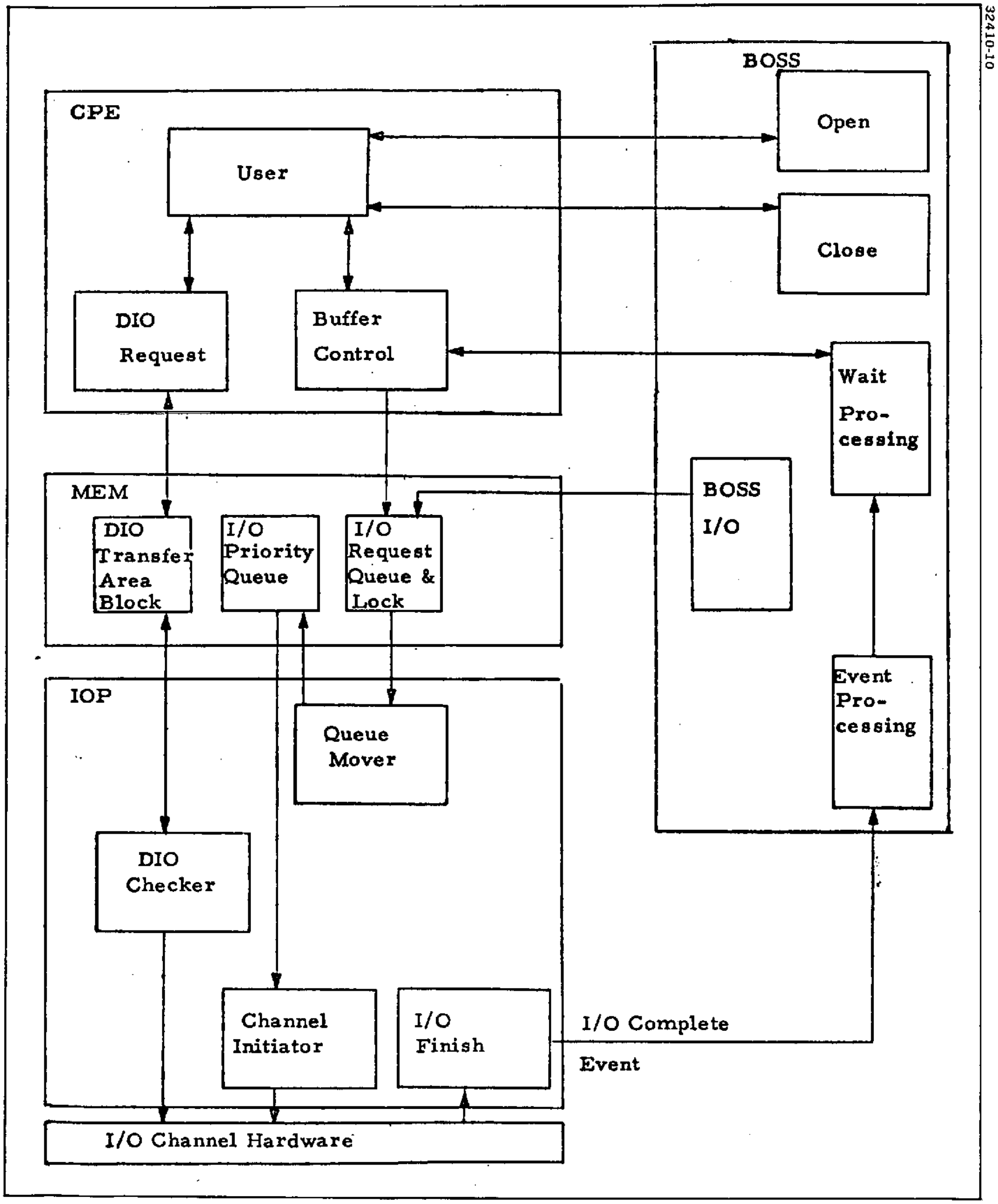


Figure 3-10. ACES I/O System

detected, Normal I/O Finish is called. If an error is detected and it can be retried, the Retry routine is called. If the fault is of a type which cannot be retried, or it has already been retried the maximum number of times (as specified by the application designer), an appropriate failure handling routine is called.

Normal I/O Finish is called for successful I/O completions. This routine marks the buffer complete, informs BOSS of the I/O completion event, purges the request from the queue, and makes the channel, bus, and device available for further use. It must also handle retry operations which require an additional operation to be performed on the device; i.e., back-spacing a magnetic tape before retrying.

The Retry routine handles error conditions. It first checks the device to determine whether a special retry routine applies. If there is one it is called. Such a special routine may specify another operation needed to clear or reset the device. If so, the old operation must be remembered and a flag set so that special handling may be provided.

Select Alternate is the routine for handling device failures. Its primary goal is to select an alternate device according to the Physical I/O Device Table and to retry the failing operation on the new device. If there are no alternate devices, the request is purged, the buffer is marked in error, and BOSS is signalled to indicate the I/O completion event of the request. Finally, the device is marked failed and the bus and channel marked available.

BOSS Routines

The Open routine allocates the logical device to the file via the Logical I/O Device Table. It allocates space for buffers. If the file is opened for input, the buffers are primed by queuing an input request for each.

The Close routine deallocates the logical device, cancels any outstanding requests, and deallocates the buffer space.

The BOSS I/O routine places BOSS I/O requests in the I/O Request Queue. This routine is similar to the CPE routine which performs the same function.

3.4.4 Service Management

Several user services and some services needed for scheduling jobs are implemented through routines residing in this layer. Before describing these functions, a brief overview of simplex/duplex memory utilization is desirable.

Simplex/Duplex Memory Utilization

In ARMMS hardware, logical pages may be either simplex or duplex. ACES software further allows any memory page to be either pageable or locked.

The pages used for ACES main memory must be duplex and locked. The pages used for application programs are configured to fit the needs of the jobs using them.

Only read-only information (constants and the code for non-self-modifying programs) should reside in simplex memory. This information can always be reloaded to its original state so it is not necessary to keep duplicated copies of it in main memory. If a task is to continue operating through memory failures, its variable data must be in duplex memory. For a critical task it may also be desirable for its non-variable data to be stored in duplex modules. This will provide somewhat higher fault coverage, and will allow the task to resume operation more quickly after a memory failure. Reloading a simplex module requires access to bulk memory, whereas duplicating the contents of the surviving module of a duplex page requires access to main memory only.

Memory used for I/O buffers must always be locked, for paging activity could seriously interfere with operation of the I/O system. It is also desirable for I/O buffers to reside in duplex memory. In some circumstances it may be impossible to recover lost I/O data which was in a failing simplex memory. The Get Main Memory routine currently is expected to obtain working storage only in a locked duplex area of memory, thus all I/O buffers should be obtained through this method to be insured of locked and duplex memory.

Direct User Services

The routines in this layer are called by the Request Processor to handle several ACES user service requests.

o Memory Allocation - Temporary working storage may be obtained by any task. This working storage is always obtained from available duplex memory. The allocatable memory area is divided into variable-size blocks of available and in-use memory. Each block has a descriptor word denoting its size. All available blocks are linked together to form a list. When a block is requested, the available list is searched for the first block of sufficient size. If one is not found, the user is informed. If a block is found, an in-use block is created from it. If the left part of the available block is larger than a certain minimum size, the block forms a new available block.

When an in-use block is returned to the system by a user, it is combined with any adjacent available blocks.

- Lock Variables - Predefined sets of contiguous data locations which must be shared between two or more independent tasks can be read locked or write locked.

  A read-lock, applied to a set of data, prevents any other task from modifying that data set until the read-lock has been removed. A write-lock, applied to a set of data, prevents any other task from reading that data set until the write-lock has been removed.

  To accomplish the locking, ACES uses "Lock-Variables". A Lock-Variable is a memory location that contains lock information pertaining to a contiguous set of shared data locations. To facilitate their use, a hierarchy of Lock-Variables may be defined.

  Since lock-variables may have a hierarchy structure, the Lock Request routine must insure that all lower level variables can be locked before any lock is actually applied. If no lower levels are found, or, if found, are of the same type, the lock request is fulfilled by setting the proper indications and incrementing a lock count in each lock level below the requesting variable involved in the lock hierarchy.

  If a dissimilar lock is found in the lock search, and the lock request cannot be fulfilled at this time, the user is informed. If the user cannot perform any useful function until the Lock-Variable is unlocked, the task may request to enter the wait state until the Lock-Variable is unlocked.

  To release a locked data set, the user makes an unlock request. The count of the number of similar locks is decremented by the unlock routine. If the count reaches zero, indicating the variable has no further lock requests, it is unlocked. Since a task may be waiting for a variable to become unlocked, the event of the variable becoming unlocked is denoted. This procedure is followed until all lower locks in the hierarchy have been processed.

- System Subroutines - In order to prevent re-entrancy problems with subroutines shared by different tasks, ACES provides a locking mechanism for shared subroutines. To call a shared subroutine, a request must be made to ACES. The System

Subroutine request service checks the lock on the requested subroutine. If it is locked, the caller is informed that the subroutine is busy. If it is not locked, it is loaded, the user's task is saved, and the CPE's executing the tasks are provided with appropriate base/bounds to access the subroutine and allowed to call it.

On exit, such shared subroutines must request that ACES remove the lock. When this is performed, the saved user's task is restored in the CPE and allowed to continue.

- <u>Partition Boundary Movement</u> - This service allows the user to adjust the sizes of partitions to allow for changing job mixes. The request is performed by changing entries in the Partitions Allocation Table.

<u>Job Scheduling Services</u>

These routines perform services required internally by ACES.

- <u>Memory Partition Allocation</u> - Jobs are loaded into the four memory partitions. When a job occupies a partition it must be allocated, and when it leaves the system, its partition must be deallocated. At allocation, the Memory Partition Table is marked, and the logical pages belonging to the partition are marked in the Logical Address Assignment Table (LAAT) as simplex or duplex, locked or not locked as determined by the needs of the job. At deallocation, the partition is marked free and its pages in the LAAT are marked free so that the physical modules they may occupy will be freed to the system.

- <u>Job Resource Comparison</u> - This service is used by Job Scheduling to determine whether sufficient resources are available to run a particular job. Currently, the only resource checked on a job basis is main memory. A partition of sufficient size must be available before a job can be run.

3.4.5 Time Management

<u>Timer Hardware Review</u>

ARMMS hardware includes two 16-bit timers of 100 μs resolution. One of these, the Real-Time Clock, counts continuously and produces an interrupt each time it overflows. At each interrupt, a software routine increments a software extension of this counter. Mission time to a resolution of 100 μs may be formed by concatenating the software and hardware portions of the clock.

The other timer is an Interval Timer. It can be set with an initial value and produces an interrupt when its count becomes zero.

Figure 3-11 depicts a conceptual view of ACES Time Management.

Timer Queue Processing

TQI's awaiting a specific time are placed in a Timer Queue which is ordered by the requested time. The Timer Queue Processor computes the interval between present time and the requested time of the first entry in the queue. If the interval cannot be contained in the 16-bit Interval Timer, a flag is set for the real-time clock interrupt processor, which will re-process it when it will fit, otherwise a full value is placed into the Interval Timer clock.

When the Interval Timer produces an interrupt, the first entry in the Timer Queue is processed and a check is made to determine if the time request has expired. If so, the TQI is moved to the Priority Queue. If not, the interval timer is loaded with the remaining interval (if less than 16-bits) or with the timer's full value.

User Timer Service

The user may read the Real-Time clock via a request to BOSS. The Real-Time clock routine reads the clock in mission time and reformats it as required by the user.

3.4.6 Scheduling Management

Several of the scheduling routines fall in this layer. Figure 3-12 depicts a conceptual view of the intertask communication of routines described herein.

Scheduling Tasks

The main Task Scheduling routine and its two principle subroutines which perform these functions are located here. For any Task Scheduling request, a TQI must be built, which requires that a TQI slot in TQM be obtained. If no slots are available in TQM, the request cannot be handled and must be rejected. The user is informed when the request cannot be handled.

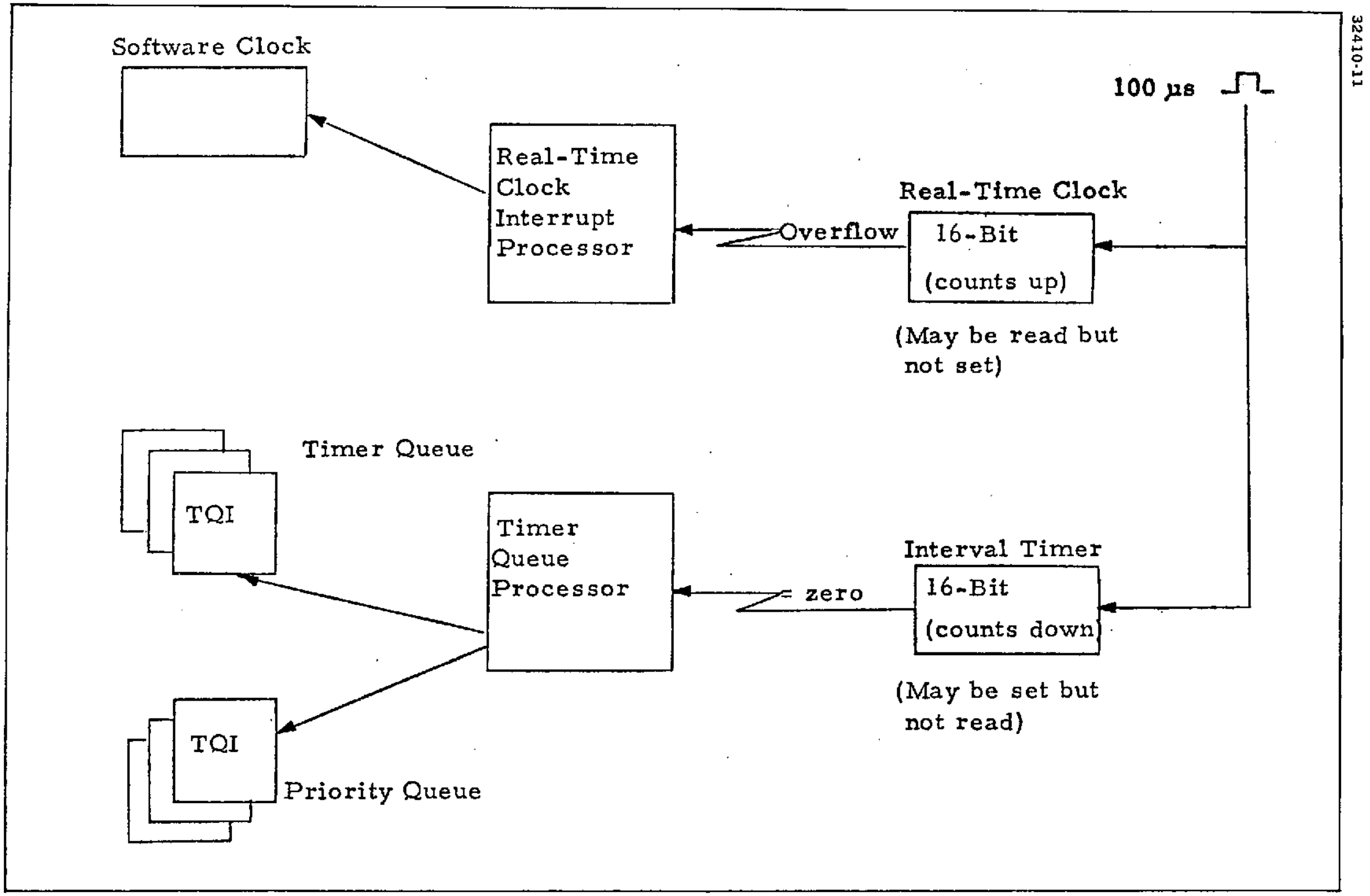


Figure 3-11. ACES Time Processing

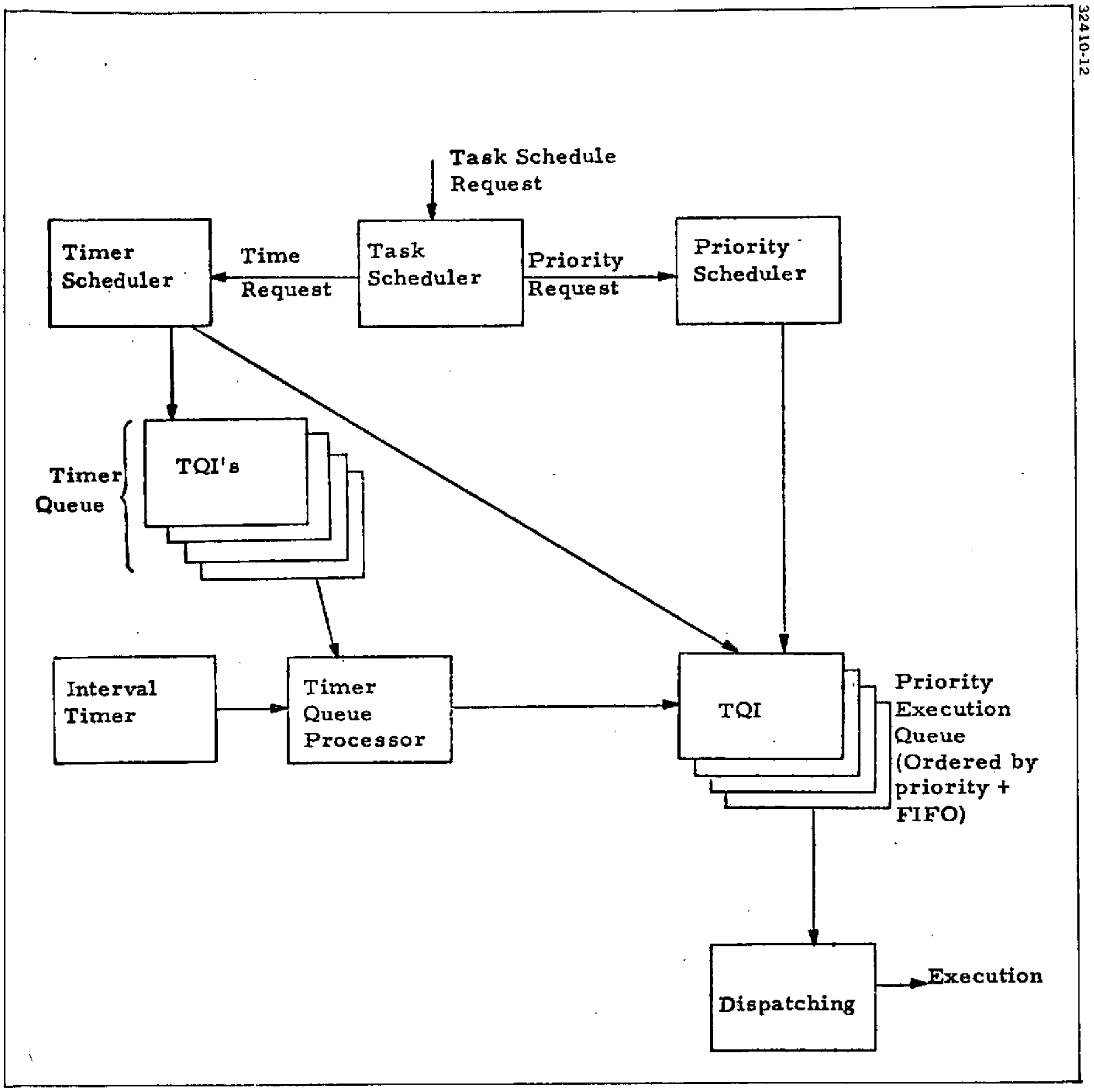


Figure 3-12. Scheduling Tasks

If any wait conditions are specified, wait items must be built, which requires space in file memory. Again if no slots are available, the request cannot be handled, and the user must be so notified.

1. Scheduling by Priority

   To schedule by priority, the TQI is built; Wait Items, if necessary, built and enabled; and the TQI entered into the Priority Execution Queue. This is done quite simply. The priority of the task is examined, the search macro-instruction is invoked to scan for the first entry in the queue with lower priority, and the insert macro-instruction is invoked to insert the new TQI before the one found by the search.

2. Scheduling by Time

   To schedule a task based on time, the TQI is built, any Wait Items built and disabled, and the TQI is entered in the proper place in the Timer Queue, which is ordered by time. If it turns out to be the first item in the queue, the Interval Timer must be reset to time the interval until time to handle this new first item. At the appropriate time, the TQI will be moved from the Timer Queue to the Priority Queue and its Wait Items (if any) enabled.

Task Wait Call

This user service routine stops a task, sets its wait bit, and initializes the Wait Items needed to make the user's task wait as requested. If there is insufficient file memory for the Wait Items, the request cannot be accepted and the user is so informed. Wait and Event processing is described more fully in Section 3. 3. 7.

3.4.7 Event Management

The ACES Event Processing system is used to control the execution of tasks based on events. Figure 3-13 presents a conceptual view of the Event Management performed by ACES.

Event Definition

An event is any occurrence which is known to ACES. Examples of events which have been defined to date include:

- Task Termination

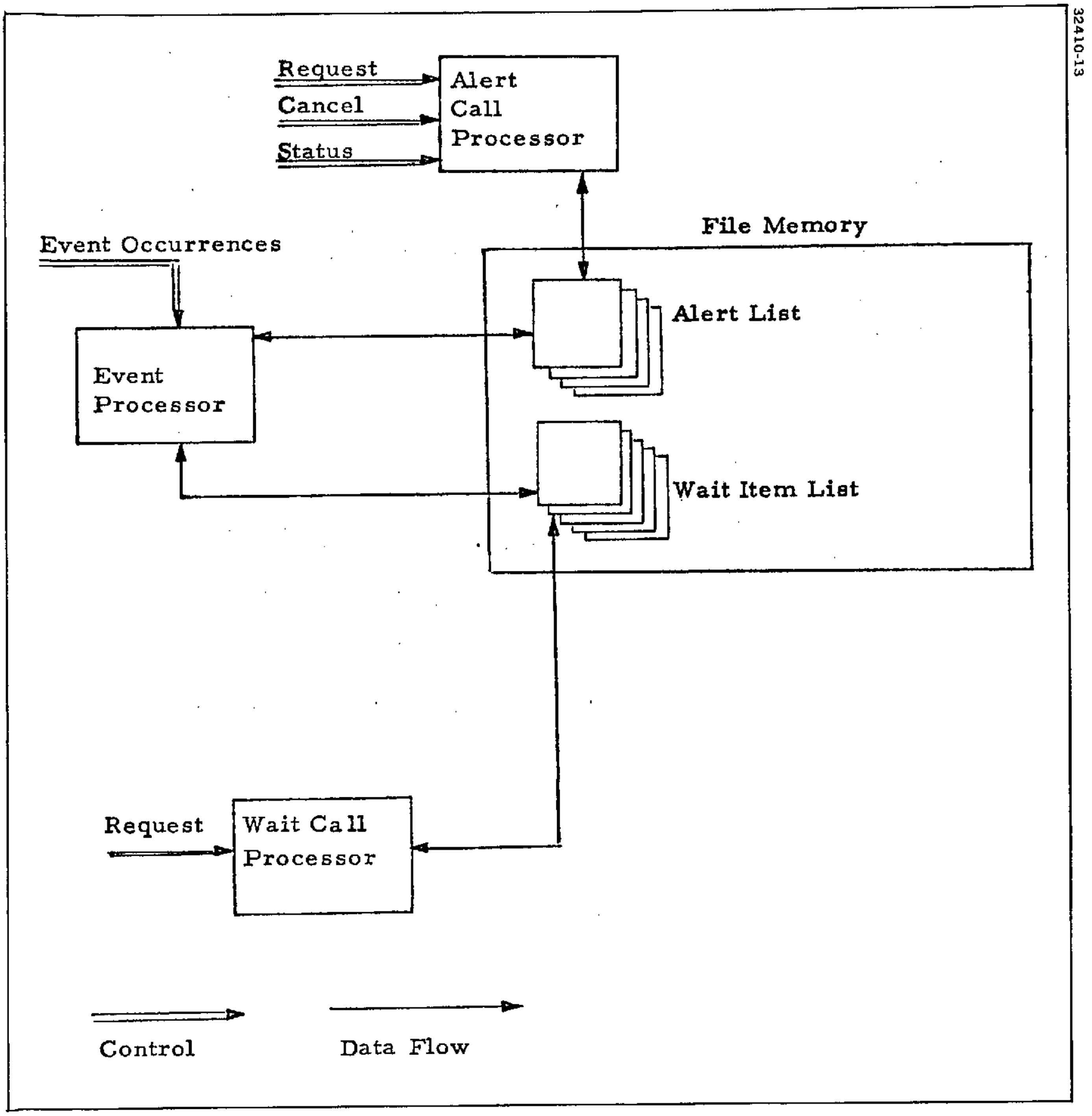


Figure 3-13. Event Management Overview

- Logical Page Available
- Variable Unlocked
- I/O Complete

Others may be defined for a particular application.

An event may be considered as a pulse. As an operating system ACES makes no attempt to remember, within itself, each event's status; ACES only responds to each occurrence of an event at the time event notification is made to ACES. However, ACES provides the user with a means of recording the status and count of event occurrences by a mechanism called Alerts.

Wait Items

A Wait Item is an entity created by ACES in response to a Wait Call request or a Task Schedule request specifying event names to be waited upon. The purpose of a Wait Item is to monitor a single event and to identify a task whose execution awaits that event. A single task may have more than one Wait Item and a single event may be monitored by more than one Wait Item. Sometimes (when both time and Wait Items are specified) Wait Items are created before the time when it is desired that they begin monitoring their events. In this case, they are built normally but they are disabled so that the Event Processor will ignore them until the time requirement has expired.

The TQI contains a counter which identifies the number of events that must be satisfied before the task may be reactivated. Each time an event occurs for which a TQI is waiting, and the event has not been previously noted by the TQI, the wait counter is decremented. When the counter reaches zero, all the TQI's Wait Items are deleted and the "wait state" status removed.

Alerts

An Alert is an entity created by ACES at the request of the user. Its purpose is to monitor a single event and to remember and count the occurrences of that event. An Alert may be substituted for an event name in any Task Schedule request or Wait Call. By controlling the time at which an Alert is created, the user may impose a wide variety of time constraints on the monitoring of events for the purposes of scheduling and waiting.

File Memory

Alerts and Wait Items are built in File Memory blocks, and linked together to form two lists: the Alert list and the Wait Item list. The File Block Status Matrix records the status (in use or available) of each block in

File Memory. When new Alerts and Wait Items are created, they are built from available blocks; when they are deleted, their blocks are returned to the spare pool.

Processing Events

Some events occur due to conditions detected internally to ACES, such as File Memory Available, ABEND, etc. Other events are detected or created by user software in the CPE's or by hardware and software in the IOP. These events are signalled to ACES through Event Set requests.

When ACES is notified of an event, it processes the event by searching File Memory. First, the list of Wait Items is searched for enabled Wait Items referencing the event which has occurred. When such an event is found, the event threshold count of the TQI it references is decremented by one. If the count reaches zero, the TQI's wait state is reset and all Wait Items referencing it deleted from the list. This search continues to the end of the list.

The Alert list is then searched, and the counts of any Alerts referencing the event are incremented by one and the event status set to complete (satisfied).

Event Based Scheduling and Waits

The event based processing of tasks for scheduling and for waits is quite similar. First, the TQI wait bit is set and Wait Items are built for all events the task is to await. Then time requests are handled. If there is a time request, the Wait Items are disabled until the requested time arrives. At that time, any of the Wait Items specifying Alerts will be initialized. The Alert is examined; if its event has occurred, the TQI's threshold count is decremented and the Wait Item deleted. Otherwise, the Wait Item is left. All of the Wait Items are then enabled.

3.4.8 Task Resource Management

This layer groups together several routines whose functions relate to management of tasks and their resources.

User Services Provided

- Task Cancel - Cancelling a task removes any pending requests for its executions and halts rescheduling if the task is periodic. This function is available as a user service request and also is used internally by ACES when a job terminates.

- <u>Task Status</u> - In order for the user to manage his tasks, it is often necessary to know another task's current status; i.e., awaiting an event, executing, pre-empted, etc. This information is recorded by ACES in the Status field of the TQI. The Task Status request allows the user to request and receive this information concerning any TQI's status.

<u>ACES Task Management Processing</u>

- <u>Task Dictionary Comparison</u> - When units have failed it may become necessary to reduce the workload of the ARMMS system. This is done by scanning each job's Task Dictionaries of Lower Level (DOLLs) to locate one which can be run on the currently available resources. The Task Dictionary Comparison routine performs the comparison needed to compare a DOLLs. needs and the currently available resources.

- <u>Job Task Halt</u> - At job termination, Job-Task Halt is invoked to apply the Cancel service to all tasks of the job. This begins the process of allowing the job to come to an orderly and timely completion.

3.4.9 Task Dictionary/Task Queue Memory Management

The Task Dictionary and the Task Queue Memory, two of the most important data structures of the ACES system belong to this layer. Most of the routines in this layer are devoted to managing these structures and providing access services to them for routines on other layers.

<u>Task Dictionary Management</u>

By calling upon the Task Dictionary Manager, ACES routines may read and write entries in the Task Dictionary in a controlled manner.

A caller may request a Task Dictionary entry for any job or job phase and read all or any part of it. Properly called, the Task Dictionary Manager will sequentially read entries from a job or job phase and provide an indication when the end of the job or job phase is reached.

A caller may also write any entry or any part of a Task Dictionary entry. An update may also be performed in which the environment will be protected between reading and writing of an entry.

Task Queue Memory Management

The Task Queue Memory (TQM) consists of many slots each of which may accommodate one Task Queue Item (TQI). TQM management is concerned with creating, destroying, reading, and writing of TQI's and linking and de-linking those TQI's into the Timer Queue and the Priority Execution Queue.

To create a new TQI, an empty TQM slot must be formed and allocated to it. When TQM is full, the caller must be notified. When a TQI is no longer needed, its slot must be made available for reuse.

It is possible to read or write all or part of a TQI. TQI's may also be read sequentially and scanned. TQI's reside in either the Timer Queue or the Priority Execution Queue. The normal sequential order for reading them is by their order in these queues. Sequential reads must specify which of these queues is to be read. The scan feature allows either of these queues to be searched for a particular TQI.

Since the Timer Queue and the Priority Execution Queue are organized slightly differently, different routines are provided for linking and delinking TQI's into the two queues. TQI's in the Timer Queue are ordered by their time parameters. At any time the Interval Timer contains the interval in 100 μs increments until time to process the first item in the queue. When a new item is placed in the queue, if it becomes the first (or only) item in the queue, the Interval Timer must be reset with the new value. The Priority Execution Queue is ordered by task priority and, within a priority, FIFO.

Task Dispatching

The two initiating routines for task dispatching interact closely with TQM so they are included at this level to allow them to access TQI's directly.

The Dispatcher routine is called to search the Priority Execution Queue for a task which may be put into execution. The Pre-dispatcher is called to perform an abbreviated check any time a task is scheduled to determine whether it is necessary to call the Dispatcher. These routines interact heavily with routines in the Initiation Management layer (Section 3.4.10).

3.4.10 Initiation Management

The routines described herein, together with the Dispatcher and Pre-dispatcher described in Section 3.4.9 form the dispatching system of ACES.

Dispatching Overview

The ACES dispatching system is presented with tasks having different priorities and stream weights. The stream weight of a task is the depth of redundancy needed by the task (1 for simplex, 2 for duplex, or 3 for TMR).

The Dispatcher has at its disposal up to four identical CPE's. These may be configured in any combination to form streams of weight 1, 2, or 3. At any one time up to four simplex, one duplex and two simplex, two duplex, or one TMR and one simplex streams may be executing. The Dispatcher selects tasks for execution, selects CPE's to execute them, configures the CPE's, and starts the tasks. When a task terminates, ABEND's, or goes into the wait state, it is a function of the dispatching system to stop the task's execution, return its CPE's, and update all tables accordingly.

Dispatching Tables

The Dispatcher uses several tables to hold information concerning the TQI's and CPE's it manipulates.

1. Priority Execution Queue - Tasks awaiting execution are kept in the Priority Execution Queue, or simply Priority Queue. Tasks (TQI's) are placed in this queue by the Priority Scheduler and remain there until they terminate, ABEND, wait for a time expiration, or are cancelled.

   This queue of TQI's is in a linked list format. It is ordered by the priority of the TQI's. When two or more TQI's have the same priority, they are further ordered on a first-in, first-out (FIFO) basis.

2. Master Execution Table - The Master Execution Table (MET) is of primary importance in ACES. It identifies each TQI currently in execution by its TQI number and keeps track of which processor(s) the task is using. It is also used to identify the active CPE's active TQI's, etc. The MET is also referenced to identify the requesting stream whenever a service request interrupt is processed by ACES.

3. AVAIL Word - The AVAIL Word is used to record the available CPE's and buses. It contains one bit for each CPE and one bit for each bus in the system. When the CPE or bus is free for use, its bit in the AVAIL word denotes its availability.

4. Configuration Resource Requirement Table - This table contains an entry for each possible combination of CPE's and buses that can be used to form a stream. Each entry is a word in the same format as the AVAIL word, each bit indicating a CPE or bus which must be available to utilize the entry's configuration. It is arranged in three columns, one each for simplex, duplex, and TMR combinations.

5. Configuration Connect Word Table - This table is arranged in the same form as the Configuration Resource Requirement Table. For each Configuration Resource Requirement Table entry, the corresponding Configuration Connect Word Table entry has the information needed to "wire" the hardware into the correct configuration.

Operation of the Dispatching System

Figure 3-14 presents a conceptual view of the operation of the Dispatching system.

1. Task Selection - The first step in dispatching a task is to select from the Priority Execution Queue the highest priority TQI not in the wait state for which sufficient resources exist. Sufficient resources exist for a TQI if either they are already available or they are in use by a task or tasks of lower priority which may be pre-empted.

   The Priority Execution Queue must be searched, beginning with the highest priority entry and continuing until it is certain that the queues contains no more dispatchable tasks. Each entry must be examined for wait state, stream weight, and priority needed to pre-empt another task.

   The SEARCH macro-instruction is used to search the queue for TQI's of suitable stream weight and not in wait state. This searching is controlled by the macro's mask.

   This mask is initially set to search for a stream weight of three or less. In other words, initially a search is made only for a TQI not in wait state. When a likely TQI candidate is found, the SEARCH macro stops. The found TQI's resource needs are then compared to the resources currently available (idle). If enough resources are available, the task is initiated upon them. If enough resources are not available, a check is made to determine if pre-emption is possible. If so, pre-emption is performed and the task initiated. If enough available resources and lower priority tasks do not exist to form a stream of the proper weight, it is an indication that a stream of this weight cannot be placed into execution at this time. Therefore, the search mask word

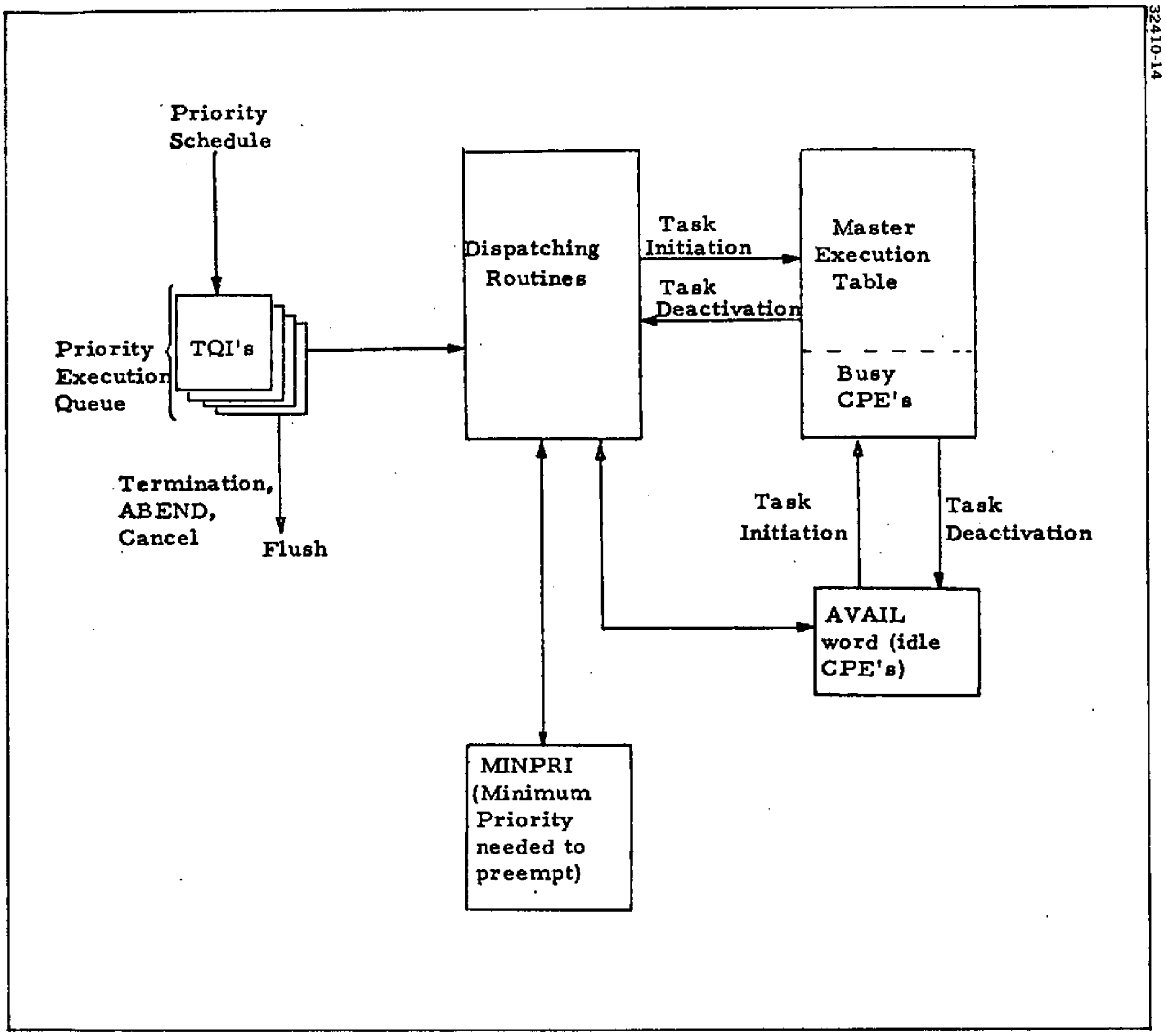


Figure 3-14. Dispatching Overview

is set to search for a task of one stream weight less than the current TQI's stream weight and the SEARCH macro is restarted at the next TQI in the Priority Queue. The search is discontinued when the end of the queue is reached or no more resources exist with which dispatching can be performed.

2. Pre-emption - Lower priority tasks are frequently pre-empted to obtain resources to run higher priority tasks. Before pre-emption of lower priority task(s) is performed, a check is made to determine if enough lower priority tasks exist to form a suitable stream. If not, no pre-emption is performed. If enough lower priority streams exist, they are halted, one by one, starting at the lowest priority task, until enough streams are available to start the new task.

3. Configuration - A suitable configuration for the task is quickly found by selecting the column of the Configuration Resource Requirement Table corresponding to the task's stream weight and searching down the table. A simple bitwise comparison with the AVAIL word tells whether an entry is suitable for the available hardware. When an entry is found, the information in the Configuration Connect Word Table plus the address of the task's save area is used to form prepare-to-start commands for each processor in the new stream. These commands are sent followed by a hardware synchronize start command, and the processors begin executing the new task in lock step.

4. Task Halt - To stop a task, the task's save area address is sent to each processor in the stream via the prepare-to-stop command. Then a hardware synchronize stop command is broadcast to the modules. The CPE's then proceed in lock step to store the processor's current state in the task's save area. This proceeds concurrently with BOSS's housekeeping operations; i.e., adjusting its various tables to reflect availability of the hardware used by the stream, etc.

3.4.11 Fault Management

Fault Processing Overview

Faults in the ARMMS system are detected by fault-checking circuitry in the various hardware modules. Each of these modules is capable of producing a distinct fault interrupt into the BOSS processor. In addition, the CPE and IOP hardware is capable of masking many faults. In these maskable fault cases, instead of interrupting the BOSS processor at the time the maskable fault occurs, a record of the fault's occurrence is saved in the processor in its Module Status Word (MSW). Then, when the next normal task service request is presented to BOSS, both the task service request and the maskable fault record are presented. Therefore, it is possible to simultaneously have a failure indication and a legitimate service request from CPE's and IOP's.

Both the fault indications and the service requests are processed in BOSS by the ACES Request Processor routine. The Request Processor routine first examines all incoming requests to determine if faults (maskable or non-maskable) have occurred. Two types of faults could have occurred. The first is due to a logical address not currently active. The second type occurs if a hardware fault was detected. In the former case the Page Fault Processor is called and in the latter, the Failure Pre-processor is called. (By calling these routines before the proper service request routine, a pseudo higher priority is assigned to fault request over normal service request.) After calling the Fault Processing routines, or if no fault indications were present, the normal service request routines are called. Figure 3-15 depicts the Fault Processing components at a functional level.

o Failure Pre-processor

The Failure Pre-processor, using the Fault Isolator determines which module or modules are at fault. These modules are then reserved and a Module Failed Word is built for the Fault Processor which will perform diagnostics and take appropriate action.

If the failure has not destroyed the integrity of the task, it is allowed to continue processing. This is achieved by making the task appear as if it had been pre-empted. If the task's integrity is questionable or destroyed, its service code is modified to an ABEND request so that the Request Processor will initiate ABEND processing for the task. If the failure is found to be in a memory module, the Memory Failure routine is called to replace the module.

o Fault Processing

The Fault Processor is a key module in the ACES diagnostics system. It:

1. performs follow-through processing for memory paging,

2. controls the testing of suspected failed modules, and

3. handles the periodic retesting of hardware modules.

The Fault Processor is responsible for follow-through processing for memory paging. Whenever I/O has begun for a memory page, the

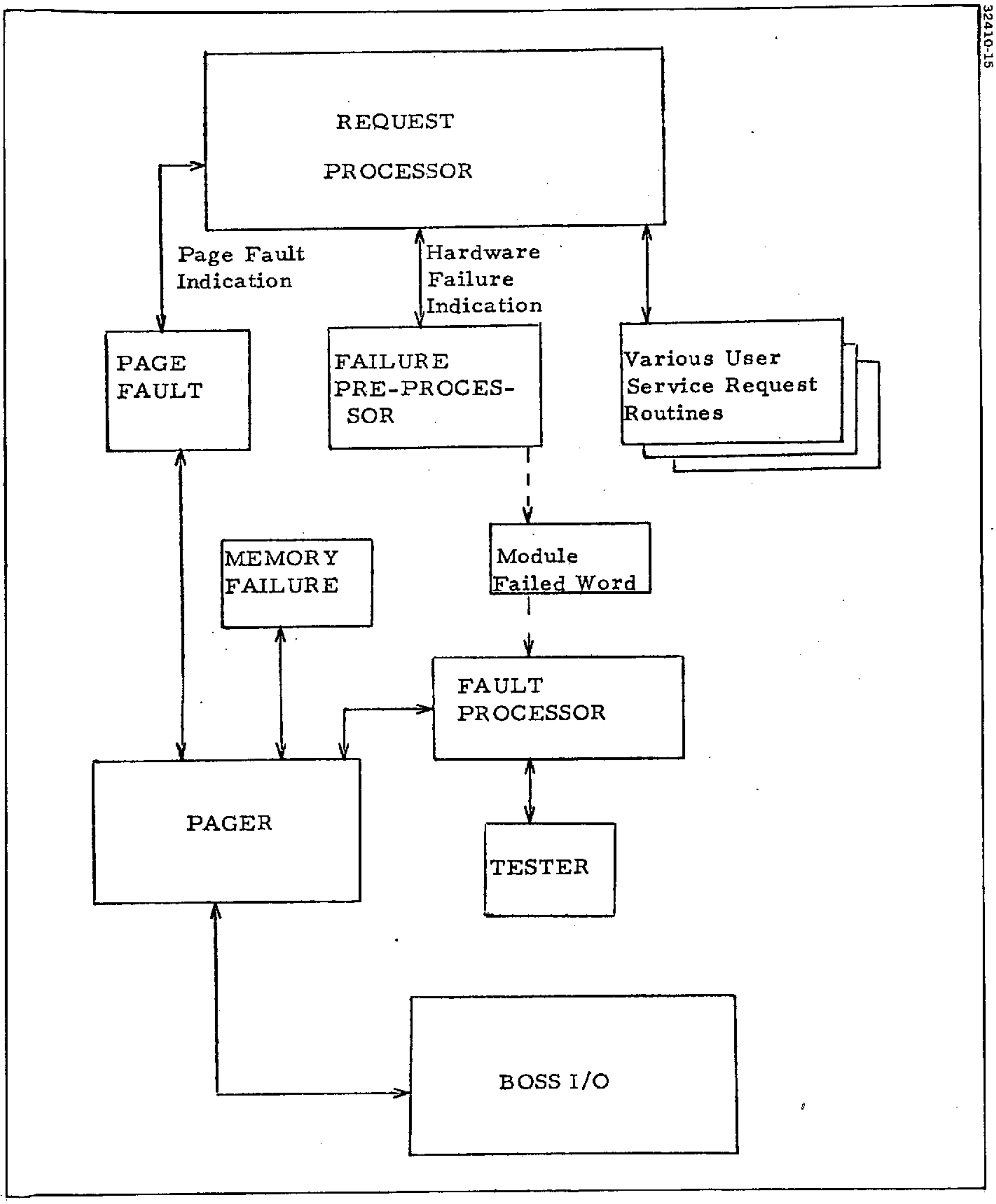


Figure 3-15. Fault Processing Components Overview

Pager routine is called periodically to check for completion of the I/O operation. If the I/O is complete, that routine performs paging complete operations.

Suspected failed modules are noted by a Module Fail Word (MFW). When modules are suspected of having failed, the Fault Processor first reserves the module for diagnostic purposes, awaits the reservation complete, and calls the Tester routine to perform diagnostic testing and replacement, if needed.

Finally, the Fault Processor is responsible for re-testing of hardware modules. Modules which have failed and been taken out of service are periodically re-tested to determine if the module has become functional again. Also, when no other diagnostic activity is present, the MSW's of all modules are scanned searching for faults which might go unreported due to a failure of the interrupt system.

- Reservation System

  The reservation system consists of two routines in the diagnostic system, Reservation Call and Reservation Return, and one routine in the dispatching system, Reservation Checker.

  Reservation Call determines if a module to be reserved is failed or spare. If either of these cases is true the module is immediately reserved for diagnostic use, otherwise it is added to the reservation request list.

  Each time the dispatching system releases a module to the system, it calls the Reservation Checker to see if a reservation request has been made for it. If it has been requested, the module is placed on the reservation list. When a reservation is active, Fault Processor periodically determines if the request has been satisfied. If so, the Tester routine, which has been waiting for the reservation, is called to perform diagnostics on the module.

  When Tester has finished with the module, it calls Reservation Return which returns the modules to their previous state, failed, spare, or operational.

Paging

ACES employs a paging scheme in its management of memory resources. Basically, any paging scheme treats memory as two separate address spaces, a logical one and a physical one. ARMMS logical address space of 128K words is divided into 16 pages of 8K words each. Each of these logical pages may occupy none, one, or two of the available physical memory modules. Non-present logical pages are stored on a bulk storage device and do not require a physical memory module(s). Present logical pages may require one or two physical memory modules according to the criticality (simplex or duplex) desired of the pages. When a task attempts to reference a non-present page, it produces an interrupt into the BOSS processor. ACES then makes that page available by reading it into main memory from bulk store.

This scheme provides two principle benefits to the ACES system: 1) A degraded mode of operation is available when failures of physical memory modules reduce their number below the number required to house the full logical address space. 2) The fault processing software may perform diagnostics on physical memory modules by removing them from their role as a logical memory.

o Page Fault Processing

When a legitimate address is referenced that is not in main memory, a page fault interrupt is generated into BOSS. In handling this page fault interrupt, ACES first places the task attempting to address the non-present logical address into the wait state to await availability of the page. It must then find a physical module(s) to house the new page. This module(s) may be found in the spares list or it may be necessary to roll another logical memory's contents out to bulk storage in order to obtain its physical module. Once a module is found, the new page is rolled in from bulk. After the new page is in main memory, the task may be removed from the wait state by ACES.

Some processes such as I/O and ACES itself cannot tolerate the delay involved in paging. It is, therefore, necessary to provide a mechanism for locking logical pages into main memory. After a logical page is locked into its physical module, it cannot be separated from that module (except of course if the, or one of the, physical module(s) fails).

o Memory Failure

When a failure is detected in a memory module, the paging system is invoked to separate the physical module from its logical page. Once this has been accomplished, failure handling of the memory module may proceed the same as for any other module; i.e., diagnostics are performed to verify the failure, and the module is marked failed or return to the system depending on the result.

Functions of System Initialization/Restart

A detailed step by step procedure for system initialization/restart has not been included herein. This is primarily due to the fact that the final hardware configuration, with its detailed list of capabilities and restrictions, has not been completed as of this writing. However, certain tables, data, etc., in ACES must be reinitialized, regardless of other requirements the hardware imposes. Therefore, an overall functional guideline is presented in the following discussion so that the system builder has an initial feel for the items that must be reinitialized.

It is assumed that hardware will provide a suitable bootstrapping procedure to configure a workable BOSS processor and ACES main memory modules and to load those modules with ACES software. To protect the software from interference, all other modules must be stopped and all interrupts must be masked. Only then can ACES software take over initialization of the system.

The flushing of active jobs, tasks, etc., is accomplished by clearing and resetting ACES tables and queues. Table 3-4 details the tables, and the states to which they must be set for ACES to be reinitialized.

The normal ACES diagnostic facilities are used to locate and establish a working hardware configuration. This is done by assuming that all modules are failed and initializing ACES tables so that hardware self-testing will begin immediately on all modules. As good modules are found, the diagnostic system will automatically update all operational tables, thus making them available immediately.

As enough modules become operational, the normal job scheduling facilities will begin loading jobs from the Job Queue, thus restarting user operations. As required by the application, ACES restart operation may:

- schedule a special restart job, or
- may flush all outstanding jobs and await manual intervention, or
- may resume processing by loading the next scheduled and available job.

After a job has been chosen for execution, ACES restart procedures will cease and normal processing will begin.

ACES INITIALIZATION REQUIREMENTS

PURPOSE	TABLE	INITIALIZATION
Clear active jobs	Job Active List Job Definition File Memory Buffers	Purge
	Job Pending Queue	(if required by application)
Clear active tasks	Master Execution Table Priority Execution List Queue Block Status Matrix Task Dictionary Task Queue Memory	Purge
Clear Waits and Alerts	File Memory File Memory Status Matrix	
Clear System Services	Subroutine Call List	
	Lock Variable Table	Remove any locks
	User Dynamic Storage Area	Set all available
Clear I/O Operations	I/O Priority Queue I/O Request Queue DIO Transfer Area	Purge

Table 3-4

ACES INITIALIZATION REQUIREMENTS (continued)

PURPOSE	TABLE	INITIALIZATION
Clear I/O Operations (continued)	DIO Lock I/O Request Lock	Remove locks
	I/O Channel Status Table I/O Bus Status Table Physical I/O Device Table Logical I/O Device Table	Set all available
Re-establish Control of Resources	Memory Module Status Table	Initialize ACES' units in use; other's failed (retest time = current time)
	Logical Address Assignment Table	ACES' addresses assigned, others unassigned
	Memory Partition Table	Set initial partition boundaries
	ACES Dynamic Storage Area	Set all available
	Unit Status Table	All units failed (retest time = current time)
Initialize Dispatching	Available Resource Word Maximum Available Stream Weight	Purge
	Minimum Preemption Priority	Maximum priority
Reset ACES Functions	Interrupt Record Module Failed Words Reservation List Communication Queue Area	Purge

Table 3-4 (continued)

3.4.12 Hardware Management

This layer is the layer which performs basic interfaces between ACES and the computer hardware. Figure 3-16 depicts an overview of the Hardware Management processing.

Interrupt Processing

The exact mix of hardware, firmware, and software for interrupt processing was not completely determined as of this document's writing. Therefore, the exact detailed functional design for the interrupting logic is not included herein. However, regardless of the exact hardware/firmware operation, certain functions must be performed in the interrupt processing logic. It is this logic that is presented here.

The ACES interrupt processing must accept notification of an interrupt and control its entry into the ACES system for processing. Most interrupts will cause the Request Processor to be executed to perform the handling of user service request. In these cases, the interrupt generating module's Module Status Word (MSW) is obtained and passed to the Request Processor for further processing.

Timers

In addition to the user's service request interrupts there are two Real-Time clocks which require frequent interrupt processing. The Interval Timer interrupt requires that the Timer Queue Processor execute and so a request is placed in the Special Request Table for its execution. The Real Time clock overflows are processed in the Interrupt Processor so that the software clocks can always be as accurate as possible.

BOSS-To-Module Bus Operations

In addition to the interrupt processing which must be performed, several routines are inclued in this layer to control BOSS communication to the external modules via the BOSS-to-Module bus. This bus transmits data to control and monitor the configuration of the ARMMS system.

One of the routines which communicates over this bus is the Read MSW routine. This routine performs the hardware interrogate command to obtain any module's MSW. Any function of ACES may call this routine to have an MSW read.

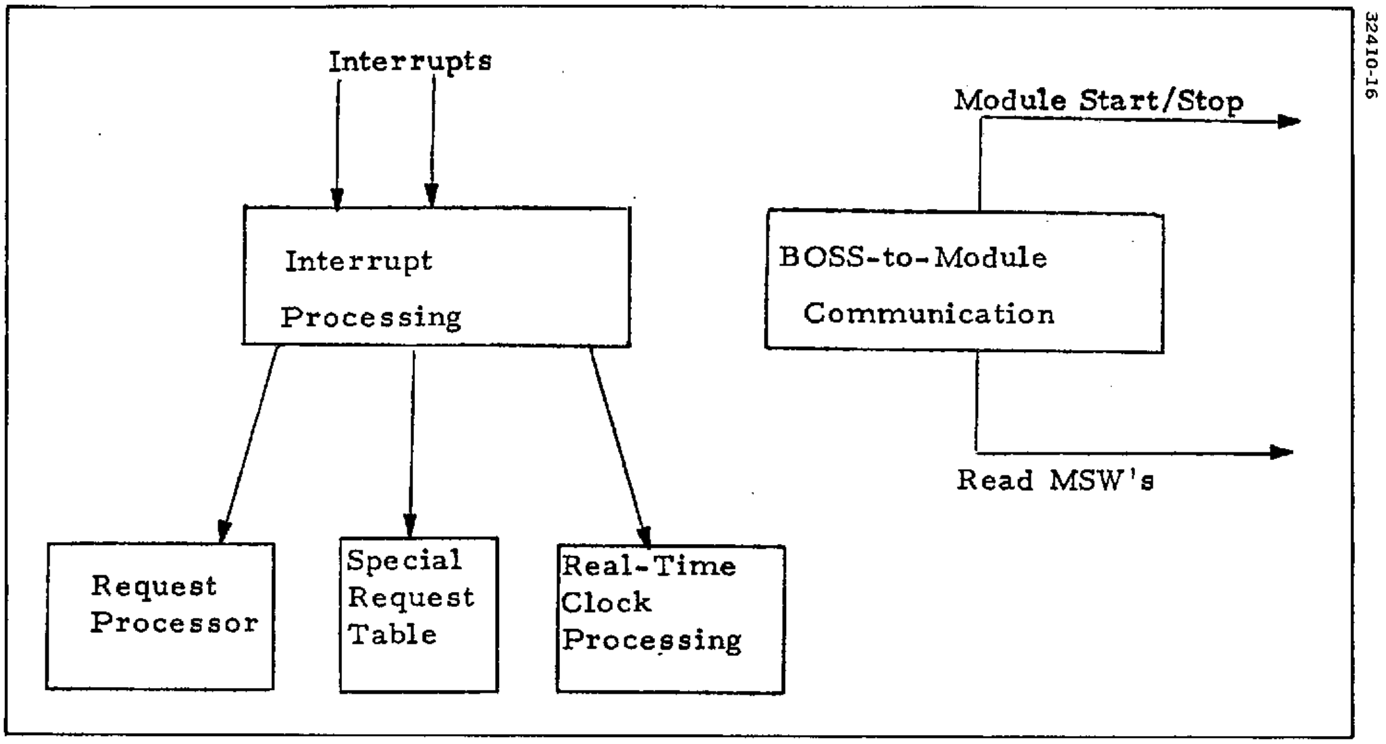


Figure 3-16. Hardware Management Overview

In addition, there are routines in this layer which control external module's "run" mode. That is, one routine, through available hardware facilities, performs the synchronize stop operation of CPE's and IOP's. Another routine performs the synchronize start operation. These routines transmit a "prepare-to-start (stop)" command to every individual processor in the stream to be started (stopped). Then, one synchronize start (stop) command is broadcast on the BOSS-to-Module bus. This procedure maintains lock-step operation for the stream. No other streams are affected by the broadcast command.

3.5 ACES Timing and Memory Utilization Estimates

Timing Requirements

The following presents timing requirements, timing estimates and detailed memory utilization estimates for the ARMMS Control Executive System (ACES). This data was generated during the detailed design of ACES. In view of the tendency for software systems to increase in size and complexity during implementation, a conscious effort was made to bias these estimates somewhat on the pessimistic side.

Early in the ARMMS project, M&S Computing assembled a set of Mission Analysis Profiles (MAP's) based on existing aerospace programs. These were used as a basis for estimating timing requirements for ACES. They guided much of the design of ACES and frequently determined the eventual structure of the system. Table 3-5 presents estimated timing requirements used to guide the ACES design.

At least two points need some clarifying comments. The "average number of ACES requests per task" is assumed to be five. This is determined by summing the average number of ACES requests per task.

1.0	Wait Request
0.5	Alert Request
1.0	Lock Request
1.0	Unlock Request
0.5	Task Schedule Request
1.0	Terminate Request
5.0	Requests per Task

ACES TIMING REQUIREMENTS

1.	Average task execution time (excluding wait time)	5 milliseconds
2.	Average number of tasks executing at any one time	2.5
3.	Average number of waits per task	1
4.	Average number of alert requests per task	.5
5.	Average number of lock requests per task (one lock also requires one unlock)	1
6.	Average percent of tasks which are periodic	40%
7.	Average time between ACES rescheduling a periodic task	5 milliseconds
8.	Average time between task schedule calls	3.5 milliseconds
9.	Average number of task schedule calls per task execution	.5
10.	Average percent of task schedule calls with wait items associated	30%
11.	Average time between task schedule calls with wait items associated	10 milliseconds
12.	Average percent of task schedule calls with time requirement	5%
13.	Average time between task schedule calls with time requirement	70 milliseconds
14.	Average number of ACES requests per task	5
15.	Average time between an ACES request per task	1 millisecond
16.	Average number of task dispatches per 5 milliseconds	7
17.	Average time between dispatches	.7 milliseconds
18.	Average time between events	.5 milliseconds
19.	Paging rate	Faults>Paging>0

Table 3-5

The "average number of task dispatches per 5 milliseconds" is assumed to be seven. Since there is an average of 2.5 tasks executing at a time, each task's average time is 5 milliseconds and each task issues one wait; there are 2.5 initial task dispatches and 2.5 re-start dispatches. This yields 5 dispatches per 5 milliseconds. It is further assumed that an average of 2 other tasks are dispatched during the time period that the tasks are waiting. This gives a total of 7 dispatches per 5 milliseconds.

One further significant fact can be drawn from Table 3-5. The average time between ACES performing a function is 200 microseconds. During a given five millisecond period there are:

7.00	Dispatches
1.25	Alert Request (.5 per task, 2.5 tasks)
5.00	Lock/Unlock Request (2 per task, 2.5 tasks)
1.25	Task Schedule Calls (.5 per task, 2.5 tasks)
14.5	

Thus, there are approximately 15 task requests per five millisecond period. The event rate is one event every .5 milliseconds or 10 events per 5 milliseconds. The 15 task requests and 10 events mean that ACES must process 25 functions during a given 5 millisecond period, or one function every 200 microseconds.

Timing Estimates for ACES Dispatcher

As the instruction set for the ARMMS BOSS processor was defined, portions of ACES were trial-coded in order to evaluate some of the instructions and to make some estimates of ACES execution speed. As study of Table 3-5 will indicate, the dispatching system is one of the most time-critical portions of ACES. It is also one of the most complicated. Therefore, it was chosen for trial coding. The results of this effort are presented in Table 3-6.

These timing estimates were also used in a simulation study conducted by Computer Sciences Corporation in Huntsville. These studies indicated that dispatching would utilize approximately 15 percent of available BOSS time, and that the Dispatcher would perform adequately under the assumed timing requirements.

ACES DISPATCHER TIMING ESTIMATES

Timings for Different Queue Structures

	Original Design*	16 Priority Levels	No PEQP**	3 Pointer Queue	3 Pointers 16 Levels	3 Pointers No PEQP	Macro-instruction
Enter item into queue	10	20	55	20	30	75	52
Delete item from queue	5	5	5	10	10	10	5
Search for dispatchable task	700	260	195	833	115	75	54
Total	715	285	255	863	155	150	111

* - See Task IV Report, M&S document 72-0027
** - PEQP = Priority Execution Queue Pointers

Timing for Complete Dispatchers Using Macro-Instruction

Routine	Time/dispatch
Dispatcher	53.8 microseconds
Starttask	23.7 microseconds
Configurator	59.2 microseconds
Tableupdate	14.4 microseconds
	151.1 microseconds

Table 3-6

Memory Requirements Estimates

Memory requirements were estimated for each routine (instructions) and table (data). Table 3-7 presents an overview of ACES memory utilization estimates. It shows the program and table memory estimates for each layer of the system. The data requirement of 5,960 words is greater than the program requirement of 4,905 words. In most operating systems data requirement far exceed program requirements. The 10,865 32-bit words insure that ACES will fit into 2 8K 32-bit word memory modules.

3.6 Design Verification

From the inception of any software system, the system designer must be constantly aware of means of verifying the completed software design. This is especially true of a spaceborne operating system such as the ARMMS Control Executive System (ACES). Design verification for ACES has been of the utmost importance throughout the entire ACES design effort and, for this reason, it is felt that ACES will be relatively easy to verify.

The following presents the means by which ACES should be verified. The subject matter is presented as a guide to help a future design verification effort flow smoothly and meaningfully.

Verification Definition

The design verification stage of a system development effort should perform three separate functions:

- Verify completeness
- Verify logic
- Project performance

Verifying a system's completeness is the first function of a design verification procedure. Verifying completeness involves insuring all necessary functions of the system are performed; i.e., all necessary software modules are present.

Verifying logic is concerned with the process of insuring those modules which are present are functioning properly. In other words, this step confirms the integrity of the system by showing that the system's software logic as presented in the design, is correct.

ACES MEMORY REQUIREMENT SUMMARY

Layer Summary	Program Memory Requirement	Table Memory Requirement	Total Memory Requirement
10. Request Management	100	110	210
9. Job Management	637	507	1144
8. I/O Management	926	76	1002
7. Service Management	705	618	1323
6. Time Management	65	0	65
5. Scheduling Management	355	0	355
4. Event Management	492	766	1258
3. Task Resource Management	140	29	169
2. Task Dictionary - Task Queue Memory	550	3517	4067
1A. Initiation Management	265	247	512
1B. Diagnostic Management	500	90	590
0. Interrupt Management	170	0	170
Total	4905	5960	10865

Table 3-7

Performance evaluation should be performed early in every system design. If performance standards cannot be attained, then time remains to change the design. Performance data must be projected with some degree of confidence for the system design to continue to the next step.

However, projecting performance is perhaps the most difficult function to perform in the design verification effort. This is primarily due to the latitude that can be experienced in obtaining performance data for a designed system. In addition, when the target computer is not yet built or readily available at this stage, as in the case in ARMMS, the problem is further complicated.

Means of Verification

At least three approaches for performing the above should be evaluated for each major area of the executive:

- o Sample coding
- o Testbed implementation
- o Simulation

Sample coding of a software component involves partially or completely coding the component. The coding should be performed utilizing the instructions available for the target computer. A sample coded program may use the entire instruction set of a machine, permitting consideration of characteristics that may be unique to the particular machine, such as addressing, special registers, etc. This can enlighten the system designer as to particular, unique characteristics that may be more fully, usefully employed throughout the entire system. In particular, data structures might undergo rigorous revision after sample coding for more efficient utilization of the instruction set.

Sample coding is best utilized as a design verification tool in the more simple, straightforward software sections. Here the sample coding is more efficiently utilized as an economical means of verifying the design. Projected performance can be adequately ascertained with non-complex software sections by sample coding. The program timing estimates are based on the manufacturer's stated execution times for the instructions that comprise the software module.

Testbed implementation is the <u>coding and execution</u> of selected portions of a software design on an actual computer. While more confidence can be placed in the results if the target computer is utilized, testbedding may be performed on any computer. For instance, if the target computer is not available; e.g., it is currently being designed or manufactured, an alternate computer may be employed.

Testbedding requires more effort to perform than sample coding. This, in part, contributes to the higher cost that should be experienced with its use. However, testbedding is more thorough than sample coding and more confidence can be placed into the verification process when it is utilized. Testbedding can be usefully employed to verify completeness, verify logic, and project performance of software systems. It is generally employed in the more sophisticated systems where sample coding is not sufficient to verify the design.

Simulation provides a testing ground for and insight into the functioning of a system and is, therefore, the most potentially powerful and flexible of the design verification techniques discussed heretofore. However, the greatest drawback of simulators is their relatively high cost.

The level of simulation to be performed is a difficult design decision. If the level of detail in the simulation is too fine, the simulator may be too expensive to use and too much machine time or capacity may be required. If the level of detail is too gross, the results may be misleading because important details may be aggregated to such an extent that their impact is lost.

Simulation provides excellent results for design verification of a new machine and software system, but the effort and cost of preparing the simulator for the full versions is usually prohibitive. Thus, for these reasons, it is usually limited to projecting performance of critical areas.

Design Verification Recommendations

The following presents the recommended approaches for verifying the different portions of the ARMMS Control Executive System. Figure 3-17 summarizes the approaches discussed herein.

1. Job Control

In the ARMMS system, a job is the highest user entity processed by ACES. A job is composed of one or more tasks which perform different, but related functions. For instance, in the space environment for which ARMMS is designed, one job might be for vehicle control, one for a life support system, while another would be for performing experiments. The vehicle control job might contain such tasks as navigation, guidance, minor loop, minor loop support, and switch selector processing.

FIGURE 3-17 ACES DESIGN VERIFICATION MEANS

	Executive Complete	Executive Correct	Expected Performance
JOB CONTROL (Job Schedule, Job Terminate, Job Cancel, Job Phase Load).	Testbed	Testbed	Simulation
TASK CONTROL (Task Schedule, Task Terminate Task Cancel, Task Status, ABEND).	Testbed	Testbed	Simulation
EVENT PROCESSING (WAIT, ALERT, EVENT).	Sample Code	Testbed	Testbed Extraction
I/O PROCESSING (File and Data Manipulation).	Testbed	Testbed	Testbed Extraction
RESOURCE CONTROL (Main Memory, Information Protection, System Subroutine, Time).	Sample Code	Testbed	Testbed Extraction

ACES job control is responsible for handling four areas:

- Job schedule,
- Job terminate,
- Job cancel, and
- Job phase load.

These areas comprise the operating system's job control processing.

Job control is one of the most difficult areas to verify in the ACES design. Although it is large, job control and task control together probably make up half of the entire ACES system for it is a very sophisticated system. Job control is a complex section whose design is ingrained in several different ACES "layers". Since the job control section is sophisticated, testbedding is recommended in order to insure that the executive is complete and correct.

To verify that the executive is complete, the job control section could be testbedded on a single processing system. However, to verify that the executive is correct, the section should be executed on a multiprocessor system. The multiprocessing system for insuring that the executive is correct is not an absolute requirement, but due to its sophisticated nature and its inter-relationship with multiprocessing, more confidence could be placed into the results if a multiprocessing system was utilized.

The multiprocessor capability would allow the ACES job control section to be executing by one processor while other processors could, simultaneously, be executing simulated tasks which make requests of job control. This would yield an environment similar to the ARMMS system where at any one time up to four processors could be making a request of job control.

At least two computer systems, located in Astrionics, lend themselves for this testbed function: the SEL 840/MP and the ARMMS Breadboard. The SEL 840/MP has three processors available. This would allow job control to be executing on one of these with the other two processors executing simulated tasks. This would permit sufficient testing to insure that the design logic is correct. If scheduling permits the ARMMS Breadboard to be assembled before design verification is complete, then that breadboard is a logical choice for verifying the design. The multiprocessing could be performed in the eventual target state of ARMMS, with BOSS and at least two CPE's.

Due to the extremely sophisticated nature of the job control section, performance can only be projected with some degree of confidence by simulation. Although simulation is costly, reliable estimates for such a complex system are probably only attainable via simulation. Testbed extraction is not an extremely accurate tool with a sophisticated system, especially one testbedded on a multiprocessor system.

2. Task Control

The ACES operating system is intended primarily to provide a reliable environment for real-time jobs. Since such jobs are generally composed of many independent tasks, considerable effort has been expended to provide a powerful, convenient system for managing such tasks. The system provides a scheduling facility which, coupled with ACES' unique dispatcher, allows the application designer to make effective use of the redundant and parallel capabilities of ARMMS hardware. Task control consists of the algorithms required to schedule, dispatch, initiate, and terminate application program tasks.

To control these facilities, ACES responds to several requests:

- o Task schedule,
- o Task terminate,
- o Abnormal end,
- o Task cancel, and
- o Task status.

The task control, like job control, section of ACES is a large and very sophisticated section. Task control is a function ingrained into several "layers" of ACES and is therefore probably the most complex portion of ACES, even more so than job control. In fact, it is the very heart of ACES with various other sections surrounding and supporting it.

Since task control is so complex, it is necessary to verify completeness and correctness by testbedding. To verify completeness, task control could be testbedded on a single processing system. However, to verify the logic's integrity, the section should be executed on a multiprocessing system. The multiprocessing system for testbedding the correctness of task control is essential. In reality, in the ARMMS system up to four processors could all at one time be making a request of task control. To insure that the sophisticated algorithms utilized are valid, a similar environment must be available during design verification. The multiprocessor testbed would allow ACES task control to be executing simulated tasks which periodically and randomly make requests.

At least two computer systems lend themselves for this testbed function: SEL 840/MP and the ARMMS Breadboard. The SEL 840/MP has three processors available. This would allow task control to be executing on one of the processors with the other two processors executing simulated tasks. This would permit sufficient testing to insure the design logic is correct. If scheduling permits the ARMMS Breadboard to be assembled before design verification is complete, then that breadboard is a logical choice for verifying the design. The multiprocessing could be performed in the eventual target state of ARMMS with BOSS and at least two CPE's.

Due to the extremely sophisticated nature of the task control section, performance can only be projected with some degree of confidence by simulation.

Testbed extraction is not accurate when a multiprocessor is utilized. While it is realized that simulation is costly, it is felt that reliable estimates of projected performance can only be attained for such a complex system by this means. Thus, simulation should be utilized for predicting expected performance for the task control section.

3. Event Processing

Event processing consists of those algorithms and design concepts required to allow application program tasks to notify the ACES of a need to monitor and record particular event histories. It also consists of those algorithms which allow the ACES to initiate certain application or system tasks in response to defined event occurrences. Typical events are specific I/O occurrences, the setting of intertask program flags, a particular task terminating, etc.

The ACES event processing system is a non-complicated system. Event processing from one operating system to another does not vary a great deal. Thus, this system over many years has been simplified very much.

Since the system is fairly simple and straightforward, it is a fairly easy task to insure that it is complete by sample coding selected portions of the system. Sample coding is an economical, yet adequate, means of verifying that this section is complete.

To insure that the event processing logic is sound, the section should be testbedded. This testbed operation can be performed on a single processing system with reliable results. This is primarily due to the mechanism, or algorithm, used to process the events. This algorithm completely processes a single event before another event can be accepted for processing. Thus, even in a multiprocessing environment, events get single processing treatment.

The expected performance of the event processing system can be adequately determined by testbed extraction. By timing the testbed operation and multiplying by a conversion timing (from the utilized computer to the target computer), fairly adequate performance projection could be obtained. This is again primarily due to the single event processing algorithm employed by ACES. The SIGMA 5 or the SEL 840 could very well be employed as a testbed computer for event processing.

4. I/O Processing

ACES contains a sophisticated, yet simple I/O processing system. This system is responsible for handling file and data manipulations between the processing elements within ARMMS and external peripheral equipment. The ACES I/O system provides two distinct I/O facilities; first, a simple, streamlined access scheme to perform I/O to real-time devices requiring only a few words of data; secondly, a more complex, multibuffering access scheme for devices requiring a transfer of many words of data. Additionally, provisions have been made available for the future addition of a FORTRAN-type format control system and/or a bulk file management system.

To verify that the I/O system's logic is complete and correct, a testbed operation is recommended. The system is too complicated for simple sample coding and not sophisticated enough to require costly simulation. This testbed operation would not have to be performed on multiprocessing computing equipment, but could easily be done on a single processing element machine.

Timing the execution of selected portions of testbed I/O system should yield some fairly reliable performance projection figures. These timings should only be concerned with actual system execution time and not with transmissional delays as these may vary greatly from one machine configuration to another.

Likely candidates for this testbed operation are the SIGMA 5 and the SEL 840/MP.

5. Resource Control

ACES provides the user with various resources needed to execute application programs. The resources may be called upon by the application task at any time during execution. Examples of the various resource control and utility programs provided by ACES are main memory management, information protection, system subroutines, and time management.

The ACES programs that exist within ARMMS are fairly small, uncomplicated resource control programs. These program designs have been kept as simple as possible during the design effort. For this reason, sample coding will prove to be an adequate, economical means of showing that the programs are complete.

To verify the designs are correct, testbed operations should be performed. These testbed operations should be small enough that each section (e. g., information protection) within resource control should be testbedded. This will insure that each independent section design logic will be verified.

The resource control program is not complicated enough to justify simulation for projecting performance. In addition, it is felt that testbed extraction should give fairly accurate performance figures.

These performance figures when transposed to the BOSS computer should be indicative of the expected performance of this system in ARMMS.

Typical computer systems, which may be applicable for testbedding resource control, are the SIGMA 5 and SEL 840. These systems should yield adequate information to determine if the system is complete, correct, and give some indication of performance to be expected.

3.7 Support Software

The Automatically Reconfigurable Modular Multiprocessor System (ARMMS), under development at the Astrionics Laboratory of Marshall Space Flight Center, offers a very flexible computing capability for a variety of space-oriented applications. To further enhance the capabilities it offers, and to make it a more cost-effective tool, support programs must be readily available to the user.

M&S Computing has reviewed support software capabilities and has established requirements for eight support software packages for ARMMS. The following documents those requirements.

General Description of Support Software

An effort was undertaken to identify support software packages which are known to be useful at existing programming facilities. From that list were selected the programs which are appropriate to include with the delivery of ARMMS to a user. The following selection criteria were applied to the identified support software packages to arrive at the list of selected packages.

- o Must provide programs which are commonly expected by a user,
- o Must provide programs which are cost-effective over manual operations,
- o Must provide programs that are unique to the characteristics of ARMMS, and
- o Must provide programs that are not commonly available on user's support facility.

In this study, each of the selected support software packages was reviewed in respect to its applicability to each of the three ARMMS processors (BOSS, IOP, CPE). This review included categorizing each support software package as required, desirable, or not required for each of the ARMMS processors. When a support software package was listed as required or desired for at least one of the three processors, this report describes the most pertinent requirements to be considered in developing the package.

Table 3-8 presents the summary of the study. At least six support software packages will be required for ARMMS' CPE. The other two processors require less support software.

However, it is interesting to note that the commonality study shows that the majority of the support packages could be developed for one processor, and with only minor or no modifications could be utilized by one or both of the other processors if desired. This should prove very cost-effective for NASA.

Currently work is being performed by the Astrionics Laboratory at Marshall Space Flight Center on the design of support software packages for SUMC. All eight of the ARMMS support software packages discussed in this report are being designed and implemented for SUMC. Some of the SUMC support software packages are due for release as early as the summer of 1973, while others are not currently planned for release until as late as winter of 1974. In general, this SUMC support software effort seems adaptable to the ARMMS requirement.

The SUMC support packages are being written to execute on almost any host computer. This is called host independency. A detailed study of the preferred host computer for each of the ARMMS support software packages was not performed. However, it is strongly felt that most, if not all, of the

TABLE 3-8 SUPPORT SOFTWARE PACKAGE SUMMARY

	BOSS	CPE	IOP
Assembler	Required	Required	Required
Macroprocessor	Somewhat Desirable	Required	Not Required
Compiler	Not Required	Required	Not Required
Link Editor	Required	Required	Desirable
Instruction Simulator	Not Required	Desirable	Not Required
Auto Flowchart	Not Required	Somewhat Desirable	Not Required
Microprogram Assembler	Required	Required	Required
Microprogram Simulator	Required	Required	Required

support software described in this study should also be written host independent. This generally implies writing the packages in higher-level languages. By being host independent, the customer should expect the fastest development time possible for the application programs, since the host computer might be changed (e.g., for compilations) to achieve the minimum turnaround possible.

Also by making the ARMMS packages host independent, it may be possible to execute some of the required support packages on ARMMS itself. For instance, it may be possible to execute the CPE's compiler on the CPE itself. Since ARMMS is designed for a high computation system it is possible that faster turnaround could be experienced with the system as a ground-based system than with other current, conventional batch systems. For example, three CPE's of an ARMMS configuration may be simultaneously compiling three separate routines.

Assembler

The assembler is the most basic support software package in use today. Without an assembler, many programs would have to be written in binary machine code. For this reason, an assembler is a required support software component for the CPE, IOP, and BOSS.

Undoubtedly, most of the application programs written for the CPE will be written in higher level languages. However, almost always where these languages are employed in a real time environment, some subroutines, segments, etc., must be written in assembly language in order to achieve required time responses. For this reason the CPE will require an assembler.

BOSS is the most time critical portion of ARMMS. Thus, a highly efficient design of ACES has been attempted. This includes a great deal of effort being applied to designing BOSS/ACES unique microinstructions. It is anticipated that ACES will be written in assembly language to take full advantage of these instructions. Also, ACES is a relatively static program which would require a new, unique compiler. A compiler unique to BOSS would not be cost-effective. Therefore, a BOSS assembler will be required.

Several of the ACES routines will be resident in the IOP. In order to meet stringent time responses, it is expected that all of these routines will be written in assembly language. This requires an IOP assembler as part of the support software needed for ARMMS.

Currently, the ARMMS system contains three distinct processors (BOSS, CPE, IOP), each with its own distinct characteristics. However, an attempt is being made to impose a similar (although not identical) instruction set upon all of them. This similarity is primarily in the instruction format. It is believed that with this similarity it is very probable that one common assembler can be developed so that it will assemble programs for any of the three ARMMS computers. A special control command could be input to the assembler specifying for which target computer is the assembly. By this means the assembler could intialize itself to the proper instruction set to be utilized.

<u>Macroprocessor</u>

As a support software package for ARMMS, the macroprocessor spans the full range of applicability from required to not required; while the package is not required for the IOP, it is desirable for BOSS, and mandatory for the CPE.

There are currently only a few IOP routines. These routines are relatively small in size. It is for this reason that a macroprocessor for the IOP is not required. There is insufficient code in the IOP to properly justify the inclusion of one in the IOP assembler. Even if one were provided for the current IOP routines, the number of times macro-instructions might be used would probably be few. Thus, it is not economically feasible to include one which requires <u>any</u> extra work in the assembler development effort.

The BOSS encompasses a somewhat larger number of routines than the IOP. These routines will be written in assembly language and could make good use of a macroprocessor if one was available. For instance, the entry/exit mechanism for ACES routines will probably require several instructions to implement. The macro-instruction capability would provide a convenient means of coding this entry/exit mechanism for each routine.

Therefore, while the macroprocessor for BOSS is desirable and could save some development cost, it is not an absolutely required support software package.

The CPE will house many thousand user routines during a mission. From mission to mission, these routines generally will experience a good deal of modification. For large, ongoing development efforts which include a fair amount of assembly language routines, the macroprocessor can save a great deal of development time by lower coding, checkout time. This service in itself makes a macroprocessor a requirement.

Also, whenever a large number of programmers are involved in a development effort, standardized procedures, such as the macroprocessor efforts, are extremely cost effective. Therefore, a macroprocessor is a required support software package for the ARMMS CPE.

All three ARMMS processors require an assembler. It was pointed out earlier in this document that it is possible to develop one assembler for all three processors and just change the instruction set according to the processor to be executing the code. With this in mind, it is felt that the macroprocessor can be also developed for the one assembler and be shared by all three processors.

Compiler

Traditionally higher-level languages afford faster coding and checkout at the expense of somewhat more inefficient code. Most compilers are only cost-efficient if they must be developed for a project, if the project is sufficiently large. For these reasons, a compiler will not be required for the IOP and BOSS, but will be required for the CPE.

The IOP will execute only a few ACES routines with each routine having stringent time response requirements. For this reason, all of the routines in the IOP will be written in assembly language. This means a compiler is not a required support software package for this ARMMS processor.

ACES is a relatively small, static program which would require a totally unique compiler if one is developed. The effort that would be required to develop the compiler would not be worth while for as small an effort as ACES.

Also, several unique microinstructions have been developed for BOSS to increase time responses needed in certain critical ACES areas. To take full advantage of these instructions, it is anticipated that ACES will be entirely written in assembly language. Thus, a compiler for BOSS will not be required as a support software package for ARMMS.

There will be literally "thousands" of CPE routines per mission. Most of these routines will be modified or completely rewritten for each mission. A compiler is ideally suited for a large programming effort where some code inefficiencies can be tolerated in order to gain increased coding and checkout time. Therefore, a compiler (or compilers) is a required support package for the ARMMS CPE.

The total commonality question for a compiler for the three ARMMS processors is almost purely theoretical. Even if a compiler was developed for the CPE which could be used by the IOP and for BOSS, it probably would not be used due to inefficiencies, even if slight, of the code produced.

It is doubtful that an excessively large portion of a CPE compiler could be utilized in the makeup of a BOSS or IOP compiler. The specialized microinstructions of the BOSS and IOP would be difficult to implement within a generalized CPE compiler framework.

Perhaps in the compiler framework "commonality" might be more applicable to the discussion of various SUMC's as being used for the ARMMS CPE's. With differing mission requirements for ARMMS, differing models of SUMC may be utilized. These differing models have differing word lengths, etc. It is plausible in this case that a common compiler for FORTRAN, say, be developed that generates code to the assembly language level only. The corresponding SUMC assembler would then be responsible for producing the machine code corresponding to the specified target computer. In this case one compiler may be common enough to satisfy all requirements.

Linkage Editor

The linkage editor allows the user to write separately assembled and/or compiled routines which can be combined to form one program, task, or job. It is responsible for "plugging" each separately assembled/compiled module together and resolving the cross references. A linkage editor is a required software package for the CPE and BOSS, and desirable for the IOP.

The ARMMS application software will probably be a combination of assembly language and higher level language routines of which several will be required to form a job. These routines will be designed and coded separately from other routines within the job. For these reasons, a linkage editor will be required to combine all routines within a job into one loadable unit.

Currently, there are approximately one hundred ACES routines which execute within BOSS. While they probably are all written in a common language (most likely, assembly), each will be designed and coded as a separate routine. Before executing in BOSS, however, they will all be required to be combined into one unit to be a linkage editor. Thus, the linkage editor support software package will be required for BOSS.

Only a few routines execute within the IOP's and each is small in size. Currently, there are no more than seven IOP routines and the combined memory requirement is less than 1K. Since the number and size of these routines are manageable, a linkage editor is not absolutely required for the IOP. It is possible to assemble each of these routines together as one large assembly. It would be the assembler's responsibility then to resolve all cross-references, etc. Since the output of the assembler is in the same format as the linkage editor, the loader should be capable of loading a module from either.

While it is not required, a linkage editor would be desirable, however. The capability to assemble each routine independently of other routines is always helpful during a debug effort. It increases checkout efficiency by not requiring all routines to be reassembled when only one routine needs updating. It is for this reason that a linkage editor is desirable for the IOP. In fact, if a linkage editor for BOSS or CPE is easily modifiable so it can handle the IOP routines, it may be well worth this effort; otherwise, a new linkage editor just for an IOP may not be cost effective.

The CPE, BOSS, and IOP routines all share common memory facilities. It is desirable, and currently planned, that all three processors share a similar instruction set and addressing scheme. These things combined with the recommendation above that the linkage editor be oriented to binary bit streams rather than consecutive fixed length words, give good reason to believe it is highly possible and extremely desirable that one linkage editor be developed for all three processors. Developing one package would yield many advantages including being highly cost effective, requiring less maintenance, and having a shorter development time.

Instruction Simulator

The instruction simulator may not be as important in the ARMMS environment as in other applications. While an instruction simulator may be desirable for the CPE's, it is expected that it will not be required for BOSS and the I/O processors.

The IOP executes only a few routines which require a very small amount of memory. It is not cost effective to develop an instruction simulator for such a small number of routines for a computer. The cost to develop the simulator would be many times over the cost of developing the IOP routines.

NASA plans call for a hardware fabrication of a BOSS early in the ARMMS breadboard phase. This hardware development precedes any software implementation. Since the processor will be developed when software implementation begins, an instruction simulator is not as important as the reverse situation where the software must be developed on a non-existent computer.

Also, the development effort for the ACES routines should not be of such a large nature as to justify the cost of the simulator. Generally, only on large development efforts, especially where limited real hardware facilities exist, can an instruction simulator be cost effective. Moreover, BOSS routines once developed are reasonably static from mission to mission meaning the simulator would only be extensively utilized for one development effort.

For these reasons, an instruction simulator is not required for the BOSS routines.

In contrast to the IOP and BOSS, the CPE has a requirement for the development of a sufficiently large number of routines to make the instruction simulator support package somewhat desirable.

The CPE will execute all of the application's routines. The number of these routines is expected to be several thousand for each mission. The large number of routines makes the simulator become useful particularly in light of the fact that most of the routines must be modified (e.g., guidance) or completely rewritten (e.g., experiment routines) from mission to mission.

However, the fact that several CPE's should be available for checkout use by the time the application programs are written, makes the simulator a luxury item and not an absolute requirement. For these reasons, an instruction simulator for the CPE is a desirable support software package for ARMMS.

While the three ARMMS processors have attempted to remain somewhat similar, they are sufficiently different hardwarewise to believe that an instruction simulator would find little commonality. For example, BOSS has an elaborate interrupt structure, while the IOP has a very simple one, and the CPE has none. Also consider that the CPE is started/stopped by BOSS, while the IOP is started/stopped by BOSS, and it is also started by a particular memory access by the CPE and by a particular I/O bus signal. Finally, BOSS is started/stopped by even more elaborate, complex hardware mechanisms. It is felt that each processor is sufficiently independent of the other two that a major redesign for each simulator would have to be performed. A simulator designed for the CPE could not be easily converted to a BOSS instruction simulator.

Automated Flowcharting System

Automated flowchart generators are the most valuable when programs are updated frequently. In such cases, up-to-date program documentation is readily available almost automatically. In relatively static programs the flowcharter serves relatively little usefulness, and in effect, "may be more trouble than it's worth". Therefore, in a system like BOSS or the IOP, a flowcharter is not required, while the CPE user may find it somewhat more desirable.

The IOP has too few routines which are static to require, or even desire, an automatic flowchart system. Therefore, an automatic flowcharter is not a required software support package for ARMMS.

The BOSS routines, once developed, are seldom altered. Therefore, the automatic flowchart system would not reach its full potential for this computer system. It is therefore reasoned that an automatic flowchart system is not a required support software package for BOSS.

Of all the ARMMS processors, the automatic flowchart system attains its most usefulness in the CPE. The application programs (executed in the CPE) will require many modifications from mission to mission and in general, the program logic is rather complex. An automatic flowcharter should produce some efficiencies for the documentation portion of each application effort. These efficiencies are not so great as to make the flowcharter a requirement, but might make it somewhat desirable. This discussion is further amplified later in this section.

The flowcharter chosen for the CPE should easily be converted to be utilized for BOSS and the IOP. The basic difference would be the introduction of the different instruction set if the flowcharter is either the syntax analyzer flowchart or chart code flowchart program type (see below). This should be a relatively simple modification. If the special language flowchart program is utilized, no modification whatever would be required as this type of flowcharter does not scan the source deck to generate the flow. In either case, it appears that one automatic flowchart system should be capable of functioning for all three systems with significant impacts.

<u>Microprogram Assembler</u>

The three microprogrammed ARMMS processors together will probably house over 175 software instructions requiring the coding of approximately 600 microinstructions.

The high frequency of execution of these microinstructions and the limited amount of microcode storage demand that this code be highly efficient. The unique nature of the ARMMS system and the difficult problems it addresses imply a relatively high probability that changes may need to be incorporated even fairly late in the design cycle.

A flexible and powerful microprogram assembler will be a required tool for all ARMMS processors to conserve programming effort while meeting these goals.

The current BOSS instruction set will require approximately 75 micro-instruction-execution routines plus several hardware interface routines to handle instruction fetch, interrupts, timers, etc. This is expected to require slightly over 200 microinstructions to implement.

As implementation progresses and experience is gained with the system, it is very likely that desirable modifications to the highly specialized instructions in the BOSS processor will be identified. A flexible micro-assembler will provide a means for incorporating such changes quickly and easily and without introducing errors in the recording process.

The BOSS and CPE processors share many hardware features in common. Many of the instructions will probably be identical in the two processors. The CPE will probably house approximately the same number of microinstructions as BOSS. Thus, to insure efficiency in code and to conserve implementation time, a CPE micro-assembler will be a required support software package for the CPE.

The IOP is as yet undefined, but is expected to have a somewhat smaller instruction set than either BOSS or the CPE. It will, however, need more microcode to handle the special hardware features supporting the IOP's special role. Coding and debugging this code will require a micro-assembler.

There is a great deal of commonality in the hardware of the three processors. The more this commonality can apply to the firmware, the easier the coding task will be and the fewer will be the sources of errors.

It should not be difficult to define a common assembly input format for all three processors. For those features which they share in common, they can also share common syntax and semantics.

There are many instructions common to BOSS and the CPE. As the IOP is designed, some of these instructions will be included. A properly designed micro-assembler could allow the same source code to be used for the same instruction in all processors. By exercising care in setting up fields and defaults, the assembler could assemble identical source statements to form properly coded micro-code for any of the machines so long as only common features (such as ACU and SPM) were used. The assembler should easily identify attempts to use features not present on the target processor.

Since the microprogram word formats are not identical, the code-generation parts of the assembler may need to be somewhat different. However, it is possible to write such routines that are largely table driven, in which case to change from one processor to another all that would be required is to load another table.

Parts of the input recognition phase of the assembler cannot be common. This includes those items referring to hardware features which are unique to a particular processor. Most error detection in this phase, however, can be common with the exception for the attempted use of features not present in the target processor.

To summarize, it seems quite feasible to implement a single micro-assembler which processes micro-code for all three ARMMS processors.

Microinstruction Simulator

Due to its intimate interaction with the hardware, timing, and other constraints, micro-code is notoriously difficult to debug on the actual target computer hardware. The complexity of the microinstructions and the requirements for extremely tight coding pose just some of the difficulties. Since the microprograms exercise and depend on the operation of every register and feature of the machine, it is potentially necessary to examine all these registers to follow microinstruction execution and locate errors. This is difficult to provide in actual hardware, specially in small, pin-limited processors such as ARMMS utilizes.

Finally, it is impossible to operate the processors until the micro-code works. Unless a simulator is provided, it will be impossible even to check out the hardware until after the micro-code has been debugged. Thus, one of the greatest advantages of a simulator will be the capability to separate debugging of firmware from hardware checkout. For these reasons, a micro-instruction simulator will be required for the CPE, IOP, and BOSS.

By having a microinstruction simulator for BOSS, its firmware check-out can be proceeding in parallel with hardware implementation and checkout. This will reduce the time required on the hardware to checkout the firmware, reducing lead time constraints.

With the number of microinstructions to be implemented in BOSS, it is more cost effective to utilize a simulator. This is due to the fact that the simulator can be executed on host computers in a batch, perhaps multi-programming, environment. When new hardware is being produced, usually only a limited amount of it is available for firmware checkout. This reduces the checkout to a sequential process.

The CPE will house approximately the same number of instructions as BOSS. For the reasons explained for BOSS, the CPE, in like manner, will also require a microinstruction simulator as a software support package.

While the IOP will probably house considerably fewer microinstructions than BOSS or the CPE, it will be the processor which must be checked out first. The IOP is the link between BOSS, memory, and the CPE to the outside world. For this reason, the IOP will be checked out before BOSS or CPE. Therefore, to facilitate this checkout in an efficient manner, a microinstruction simulator will be a required software support package for the IOP.

Although many features differ, there is a great deal of commonality in the three ARMMS processors since the basic cycle of processors is very similar.

The simulator itself will probably be different for all three processors, but its logic flow can be expected to be very similar for all three. It appears that the input to the simulator can be identical for all three processors.

Internal error checking will differ somewhat, but much of it (e.g., memory timing constraints) can be identical. The differences will be mostly due to features present in one processor and not in another.

The output for the IOP will probably need to be formatted quite differently from that for the CPE and BOSS. Those two processors, however, will probably be able to share a common output section with only trivial differences.

It will probably not be feasible to implement a single simulator to handle all three processors, but the three simulators should have sufficient commonality among them to share a considerable amount of usable logic.

SECTION 4

ARMMS HARDWARE DESIGN

This section begins with a summary of ARMMS hardware design tradeoffs and guiding assumptions made prior to phase III effecting the final ARMMS register level design. These tradeoffs include choices of operating modes, executive function location, modular partitioning, memory hierarchy, fault tolerance approach, and configuration architecture.

ARMMS has an overall reliability goal of 0.99 probability of successful operation over a 5 year mission. Register level designs and reliability analyses based upon these designs identifying potential failure modes and methods for detecting and/or masking them are given for each ARMMS module in the next topics of this section. Failure rate estimates are given for each module allowing the computation of the reliability of any desired system configurations, using the methods of section 6, as requirements for missions to which ARMMS is potentially applicable become better defined. For example a typical configuration having ten 8192 bit memory modules, five CPE, 4 IOP, and an internally partitioned BOSS module would have a probability of surviving a 5 year mission with at least 7 operational memories, 3 operational CPE, 2 operational IOP, and an operational BOSS of 0.9976. This illustrates 2 things; first, that some degradation is likely to occur and the design must cope with this gracefully and second that degradation to a single simplex stream is a pessimistic assumption except for very small initial configurations. The more likely end point is a system with perhaps two thirds of its resources still operational. The level of detail of the module design also permits descriptions of microprogram and scratchpad memory organizations, integrated circuit partitioning estimates, and proposed instruction sets.

The final three topics in this section cover tradeoffs requested by MSFC in order to bring ARMMS closer to the requirements of present SUMC related programs and known near term missions to which ARMMS is believed to be applicable. The first describes modifications to SUMC to allow its use as an ARMMS CPE providing an alternate to the other CPE design included in this section. The second describes a BOSS-less version of ARMMS aimed at adapting it to missions that could not afford or justify a full ARMMS system. The last summarizes the technical aspects of an ARMS (ARMMS with no multiprocessing capabilities) breadboard based on ARMMS design principles modified as described in these previous two subsections. The breadboard will be implemented at Hughes during 1974.

4.1 Summary of ARMMS Hardware Design Prior to Phase III

Several important trade-offs and guiding assumptions were made during the first two phases of the ARMMS study which effected the detailed design of Phase III. These are summarized in this section.

4.1.1 ARMMS Operating Modes

Three basic operating modes exist in ARMMS: TMR in which throughput capacity is sacrificed to yield highest reliability; simplex in which the converse tradeoff is made; and duplex which provides a satisfactory compromize between these two objectives in cases where all errors must be detected but need not be immediately corrected.

ARMMS modes are characterized as follows: Most faults are detected in the simplex mode but no processor faults and only a portion of those in the memory are masked. Duplex operation guarantees that virtually all faults will be detected avoiding erroneous computations but only those faults also detectable in simplex can result in masking and replacement of faulty modules with spares. The masking property means that the computer is able to complete programs already in progress before switching in a spare just as in the TMR case and that it can continue to operate in the presence of a maskable fault once available spares have been exhausted until ARMMS is commanded to change to a configuration requiring fewer active modules. Finally the TMR operation masks virtually all errors through voting. All modes have distinct characteristics which distinguish them from one another except in the special case where all modules internal error detection coverage approaches unity making duplex operation equivalent to TMR operation in performance. However, unity coverage in the processor modules results in excessive complexity for these modules in the ARMMS context and in incompatibility with existing SUMC logic and hence is not recommended.

Multiprocessing is assumed to be allowed in connection with all of these modes so long as adequate numbers of operational processors and memories are available. For a given number of modules of a given type there are a large number of submodes which could be identified. For example, if 4 processors are available they could be connected as follows:

1. One TMR machine
2. Two duplex machines
3. Four simplex machines
4. One TMR plus one simplex machine
5. One duplex plus two simplex machines

Larger numbers of processors could allow even wider ranges of configurations. However, both from the hardware viewpoint of interconnections and the software viewpoint of configuration control, some limitations must be accepted eliminating those degrees of flexibility which cannot reasonably be envisioned as requirements or those most costly in terms of hardware and software design. Four processors in use at a time seem to be an optimum maximum number since no mission requirements for multiple TMR streams have been established and 4 processors allows higher throughput to be achieved by going from TMR to double duplex operation, or by supporting a simplex mode simultaneously with a TMR mode. A capability for more than 3 processing streams operating at a time has not been clearly established. However, from a reliability standpoint any processor should be able to perform any role in any submode. For the submodes discussed only four sets of buses between memories and processors are required keeping intermodule connections and voter/switch complexity within reasonable limits. Software should be able to support these configurations without excessive complexities or operating delays. It should be noted that additional processors could be added as spares if desired with the number ultimately decided on a basis of total hardware and software costs together with reliability requirements.

4.1.2 Location of ARMMS Executive Functions

Tradeoffs were conducted during phase II concerning the retention of a dedicated Block Organizer and System Scheduler (BOSS) module in ARMMS vs. taking a floating executive approach. There are a number of reasons why a floating executive might be attractive: 1) since all processors can perform executive functions pooling of spares is made more efficient; 2) one less module type requires development; 3) if executive software overhead approaches or exceeds the capacity of a dedicated executive total processing efficiency can be lower than that of a floating concept since with a floating executive different processors may in fact simultaneously execute different executive functions. However the preponderance of evidence in ARMMS has led to the retention of the dedicated executive approach: 1) the development cost for BOSS is counterbalanced by a decrease in complexity and reliability required for all other processors and therefore which approach is more costly is not clearcut, 2) the problems associated with a processor reassigning its mode roll concurrent with monitoring all other modules would be difficult to resolve, 3) functions such as synchronization, power control, disaster restart, and interrupt reception are not amenable to distribution among processors and might require centralization in an additional module if they did not reside in a BOSS, 4) simulation efforts indicate that total executive overhead should be sufficiently low to minimize queueing inefficiencies at the BOSS interface for the configuration planned for ARMMS.

A study of BOSS/CPE commonality indicated that CPE floating point and multiply-divide functions would not be required of BOSS and that BOSS monitoring and control, timer, and interrupt handling functions would not be required in the CPE. It was concluded that despite their similarity, BOSS and CPE modules should not be made identical because of the wasted non-common logic involved (15%), the increased intermodule switch complexity if any module is allowed to assume either BOSS or CPE status, and the physical problems of inter-connecting status and control lines between all CPE/BOSS modules in a compact structure. Further, BOSS should physically be one module with several (probably 4) identically partitioned parts, any combination of which can be operated in TMR or in duplex in the event of failure of all but 2 of the parts. This will allow maximum packaging efficiency on the assumption that each BOSS partition will contain 31 LSICs and BOSS overall may have nearly 300 interconnects to other ARMMS modules. It is strongly recommended that an effort be made to maximize logic commonality between BOSS and CPE LSICs to minimize system development costs.

A study was made of a BOSS-less version of ARMMS during phase III. Its conclusions were that without a dedicated BOSS module either ARMMS multiprocessing or reconfiguration (simplex, duplex, TMR) requirements would have to be dropped and that even then a much simplified "mini-BOSS" core would have to be retained for functions not amenable to distribution among CPEs as noted above.

4.1.3 Location of ARMMS Voter/Switches

ARMMS memory and processor modules are connected by means of a system of buses and voter/switches. During Phase II of ARMMS a study was made to determine the optimum placement of the voter switches - either as additional self-contained modules external to the memories and processors or internal to the memories and processors. The study involved development and execution of a computer program to determine overall ARMMS reliability over a wide range of parameters. For the range of configurations, mission durations, and module failure rates anticipated (i.e., less than 10^{-5} failures/hour), voter placement has no significant effect on system reliability. Factors favorable to external voters are 1) a small increase in reliability, 2) a net reduction in hardware for large numbers of memory modules, and 3) increased modularity. The factors favorable to internal voters are 1) lower system pin counts, 2) elimination of the external voter module class, 3) reduction in the number of buses, 4) increased bus speed, 5) reduced BOSS complexity, and 6) reduced system power. The tangible factors favoring internal voters are considered to be more important than any small reliability loss involved – particularly since numbers of buses and pins were not reflected in these reliability calculations, the specific requirement for the marginal added reliability may not exist and moreover the difference could be removed at the system level through the use of additional memory or processor modules. Therefore voters located internally to ARMMS modules at their inputs are recommended.

4.1.4 ARMMS Module Partitioning

A study was made of processor partitioning during phase I in order to determine if such partitioning was necessary or desirable to achieve ARMMS system reliability goals. Existing processors such as IBM-MARCS, NASA-MCB, and JPL-STAR all take a functional approach to partitioning – i.e., horizontal partitioning. Raytheon's SERF computer takes a vertical partitioning approach in addition to horizontal partitioning between control and arithmetic functions. Both STAR and SERF employ internal redundancy in key portions of the control logic in addition to partitioning. The MARCs computer contains 3 functional partitions performing functions to be required of the ARMMS processor, the MCB and SERF contain 2, and the STAR contains 5, however these computer projects assumed higher component failure rates than does ARMMS because of their earlier design time frame and hence tend to be overly conservative for the ARMMS context.

The advantage of vertical over horizontal partitioning is that since all sub-partitions are identical so as long as any n of them in an n partition module are operational, a working processor can be configured. If the processor were functionally partitioned a working processor could not be configured if all of one type of partition failed even if several of another type remain operational. It is also more probable that one type will fail before others if they are not identical since there is bound to be some unbalance in the design.

The disadvantage of vertical over horizontal partitioning is that in order to attain identical subpartitions there must be an undesirable repetition of control functions as well as special controls to identify a partition function at any time. This in turn increases partition logic complexity and computer switching, and consequently BOSS hardware and software function associated with configuration control.

Internal redundancy can be used to advantage if a particular section of a module has a higher failure rate than other, or must have a higher reliability than other sections, or if a section is not amenable to error detection or correction by other means such as coding.

Consideration was given to functionally partitioning the ARMMS CPE into 2 parts – a control unit and an arithmetic-register unit. However for the complexity and consequent failure rate expected for this module this would not appear to be necessary to achieve system reliability goals. Instead use of internal error detecting codes and selective internal redundancy is recommended since this simplifies the configuration requirements on BOSS since the processor can be treated as a single unit.

A similar adjustment holds for ARMMS memory modules. Here partitioning could have been introduced into the electronics effecting single bits. However equivalent reliability enhancement can be achieved through the use of a single error correcting code, again without increasing BOSS configuration control requirements.

4.1.5 ARMMS Memory Hierarchy

Although the trend today is toward increasingly sophisticated memory hierarchies for high performance general purpose computers the weight of the evidence for ARMMS is in favor of including a small local store scratchpad memory in each processor such as was done in SUMC rather than the inclusion of a larger task or cache memory. Task and cache memories are used to allow faster access to most commonly used data than would be possible with it stored in main memory, providing a total speed close to that which could be achieved if all of memory were high speed. Typical speed ratios used are on the order of 10 to 1 for the two memory types to maximize the performance to cost ratio for the system. However these objectives are questionable with respect to spaceborne multiprocessors in general and ARMMS in particular for 4 reasons: 1) there should be no high cost ratio between plated wire and semiconductor memories (the primary ARMMS candidates) of flight rated quality; 2) the ARMMS CPE based on SUMC architecture will not exhibit a significant increase in speed while accessing a semiconductor memory vs. a plated wire memory; 3) ARMMS high packaging density should minimize propagation delays to and from main memory; 4) the sheer addition of devices, connections, watts, cubic inches and failures per hour implied by large cache or task memories is not compatible with ARMMS reliability objectives. This penalty is particularly large in the case of a multiprocessor where the ratio of task memories to processors is at least one-to-one.

A prime source of inefficiency in multiprocessing is contention for main memory access by the processors. Use of a local store with general registers allows intermediate operands to be retained internal to the processor. These registers can also be used to retain frequently used data if software is written to take advantage of this. Since a local store will not significantly increase processor complexity or significantly complicate BOSS handling of interrupts or processor faults one has been included in the ARMMS CPE.

4.1.6 ARMMS Fault Tolerance Approach

ARMMS achieves fault tolerance through voting and/or comparing the outputs of redundant memory and processor modules, the replacement of faulty modules with spares under control of the BOSS, the use of error detecting and correcting codes, and the use of selective internal redundancy within modules. Fault isolation techniques within individual modules are described in the sections devoted to those modules. A general discussion for the ARMMS system as a whole follows. (See Figure 1.)

During the ARMMS study trade-offs were conducted between differing error coding techniques. The two most promising codes considered were the residue code, which is not destroyed by arithmetic operations but does not correct errors, and the combination of Hamming plus parity codes which can correct a single error, protecting against the dominant main memory failure mode, but are destroyed by arithmetic operations. Both codes can detect multiple errors. A trade-off must be made between duplication of ALUs to detect their errors in the simplex mode and providing additional spare memory modules to compensate for the increased failure rates if no bit errors can be corrected. Since the memories have higher anticipated failure rates than the processor modules do and, if a residue code is to be internally generated in each processor and no speed penalty is to be allowed for this process, at least as much residue coder logic is required as for duplicating the processor's ALU and comparing outputs while the Hamming code is comparitively simple to generate, the Hamming code with duplicated processor ALU's is recommended for use in ARMMS.

Six code bits are required for single-error correction of 32-bit words using a Hamming code. If an additional overall parity bit is used all odd numbers of errors will be detected and the combination of these two codes will detect up to 3 errors and 50% of combinations involving more than 3 errors.

It has been determined that the simplest voter/switch design would pass data to a code checker and registers in the simplex mode, compare data bit-by-bit outputting "1" to the code checker and registers in a duplex mode, and vote on the data in the TMR mode. This requires only one holding register and one code checker per module. It masks single bit errors in all modes, and "no output" and multiple "Stuck on "0" errors in all but the simplex mode, while detecting single bit, no output, and many multiple bit errors in all modes. In duplex or TMR operation, if 2 processors both show a data error this places the blame on the memory. If only one shows an error blame is placed on the processor showing the error and its output is set identically to "0" for that operation in which case the memory module's voter/switch will accept the output of the good module as noted above.

Most error code logic resides in the processor modules. Errors are detected and corrected at the ALU input and data is encoded at the ALU output. Error detection and correction can be implemented at a cost of under 4 LSICs (250 gates each) per processor. This is approximately the same amount of logic

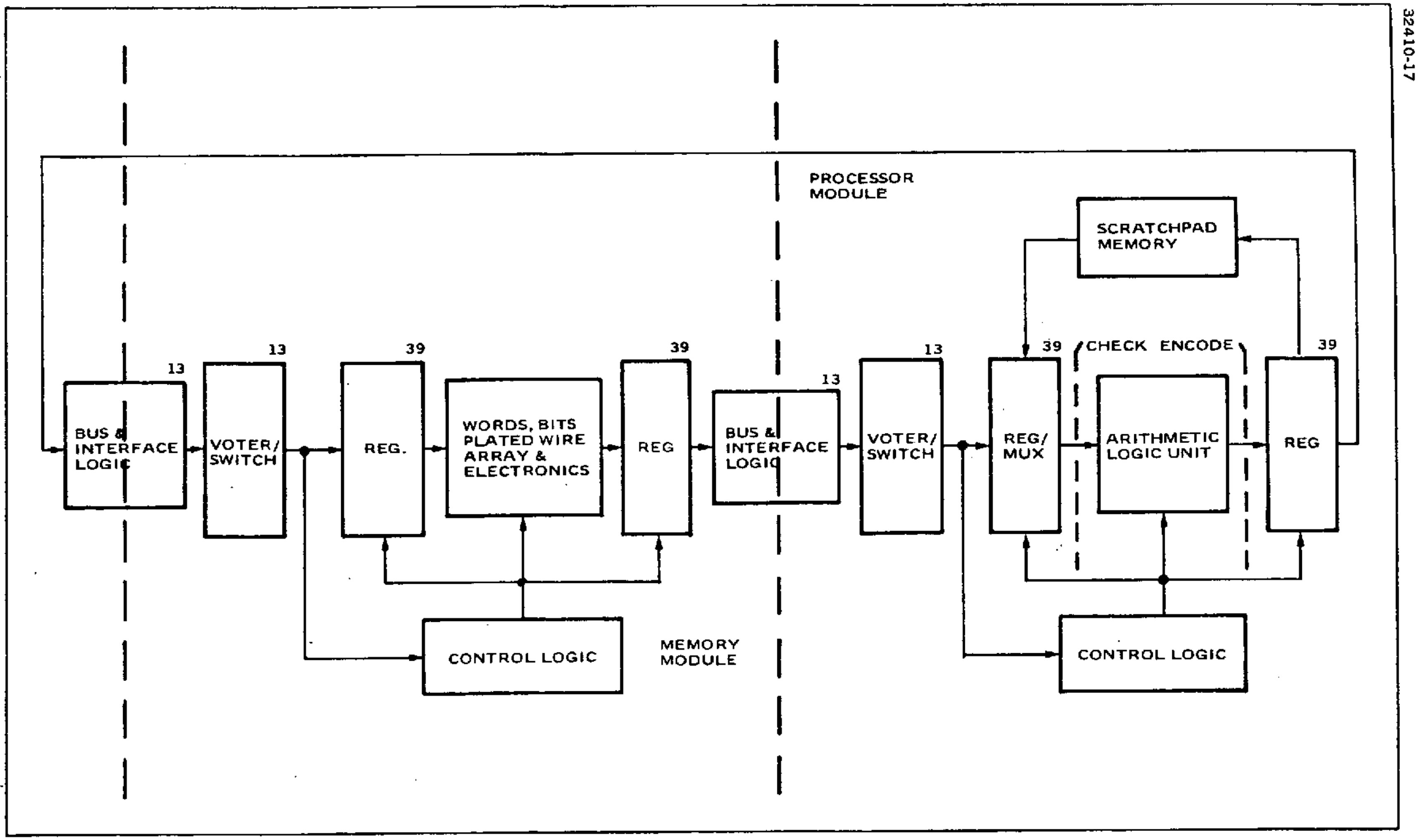


Figure 1. ARMMS Data Path for Error Analysis

that would have been required to implement a residue code checker and about twice what would have been required for parity checker plus voting.

Modules will first try to detect and correct errors by masking in the TMR or duplex mode or rollback and retry methods in simplex mode or in duplex mode cases where masking cannot be achieved. In both cases errors will be tallied. If the modules are not successful in correcting the error BOSS will be interrupted and will obtain status information from the modules in question via the BMB lines. BOSS will determine which module has failed through diagnostic routines, place it at the bottom of that module classes' spare queue and try other modules until a good one (hopefully) is found, place the good module on line, and resume computation. In the TMR mode the task will continue to completion at top priority and then the diagnostic procedure will be applied. ARMMS will be considered to have failed if and only if BOSS cannot find a usable module in each class by this procedure or if an erroneous computation goes undetected. A module is not considered to have failed until the failure manifests itself. Using internal error detection within modules allows masking of errors in duplex mode and detecting them in simplex so as error detection coverage approaches unity duplex operation looks like TMR and simplex looks like duplex. In many cases this could allow higher throughput and longer system life due to using fewer modules per stream.

4.1.7 ARMMS Configuration

One of the toughest challenges ARMMS faces is rapid reliable reconfiguration at a reasonable cost in power, volume, and complexity. Three major configurations were discussed in the ARMMS Phase II report. In addition a fourth BOSS-less configuration is described later in this report. A prime consideration of the ARMMS baseline configuration adopted in Phase II was the minimization of the number of module classes and the number of system level interconnections between modules without sacrificing reliability or performance. To this end many busses and ports of earlier configurations were combined or eliminated, memory functions centralized, and voter/switches placed internal to the modules. Four module classes and 3 internal bus classes remain; interconnected as shown in Figure 2.

BOSS – This single, subpartitioned module will execute routines for data and I/O scheduling, interrupt processing, system test, repair, and configuration, and power and clock switching and distribution. BOSS will be an internally redundant self testing and repairing special purpose computer including such instructions as LOAD, STORE, NO OP, JUMP, TEST, SPCJ, AND, OR, SHIFT, ADD, SUB, plus macro instructions to speed up frequently used processes such as table searches and special control instructions used for monitoring and controlling other ARMMS modules. BOSS will consist of four or five identical subpartitions "B" containing power supply, timing oscillator, memory bus interface and control bus voting components.

IOP – ARMMS can accommodate up to 4 I/O processors. Each I/O processor contains standard logic matching it to ARMMS system interfaces. Internally the processors can be mission dependent containing either general or special purpose logic. IOP functions include paging between bulk and main memory modules, spacecraft status monitoring and preprocessing, and spacecraft control. IOPs can be used singly, in pairs, or in triads, or can be internally redundant with multiple bus outputs.

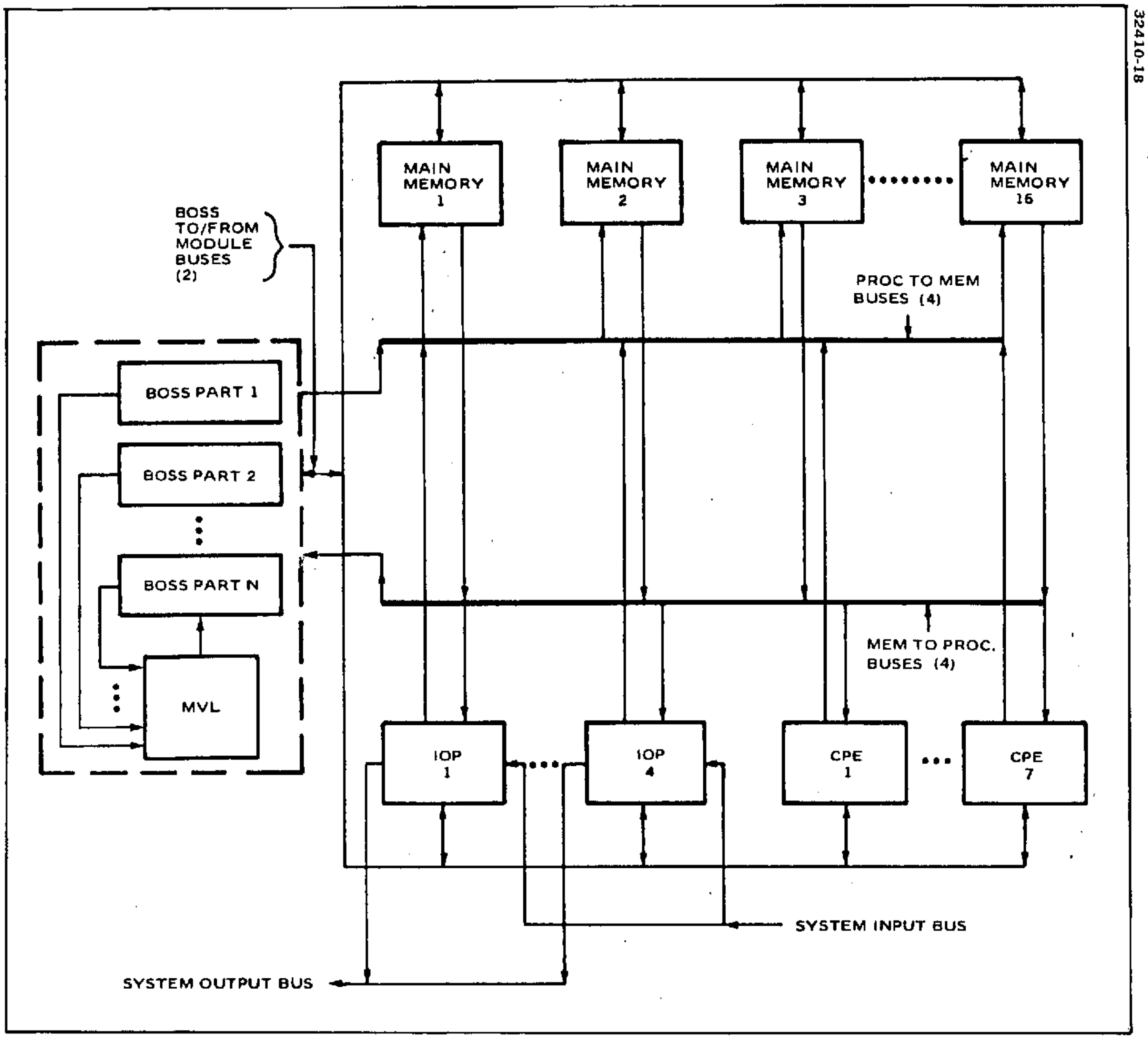


Figure 2. ARMMS System Configuration

CPE – ARMMS can accommodate up to 7 CPEs (Central Processing Elements). Up to 4 CPEs can be on line simultaneously with up to 4 IOPs and BOSS. CPEs can be utilized singly, in pairs, or in triads depending upon mission requirements. The CPE is an outgrowth of the SUMC processor modified to include self test logic, BOSS monitor and control interfaces and overlapped memory accessing.

MM – ARMMS can accommodate up to 16 main memory pages corresponding to 16 active memory modules in simplex configurations or larger numbers in dual or triad configurations. The total number of modules would be limited by bus driving components and might nominally be 25. The nominal module size is 8,192 words each containing 32 bits of data plus a 7 bit single error correcting, multiple error detecting Hamming plus parity code for data.

PMB – ARMMS contains 4 processor to memory busses. Each CPE is connected to 2 of these busses. IOPs are also nominally connected to 2 PMBs but can be connected to any number depending upon their design. BOSS and all memories will be attached to each of these 4 busses. Each bus contains 13 data lines, including error coding, and an Access request line. Software will keep track of the 2 non-existent bus ports on each processor in the same way as it does failed bus ports.

MPB – ARMMS contains 4 memory to processor busses each of which is connected to every processor module in order to allow TMR voting between any triad of busses and unlimited choice of processors with which to make up the triad. Each bus contains 13 data lines, and a response line.

BMB – Finally ARMMS contains 2 (one plus a spare) BOSS to/from module busses on which BOSS sends control codes to processors and memories and receives status information upon request. All commands and responses are coded and commands are address-tagged on this bus. The bus will nominally consist of 8 data lines plus dedicated parity, clock, and sync lines. BOSS may command or interrogate other modules at will or in response to individual interrupts from them.

An intermodule interface has been designed that allows any CPE, IOP, or BOSS module to address any non-protected memory page. It's design and operation were described in detail in the ARMMS Phase II report. It allows any combination of simplex, duplex, or TMR streams with any combination of relative priorities to co-exist with minimum bus contention providing that no more than 4 CPEs, 4 IOPs, and BOSS are involved simultaneously. The interface allows all modules of a class (CPE, Memory, etc.) to be virtually identical. Interface gate complexity and module to module interconnections have been minimized. Whenever a stream is formed BOSS sends each processor module involved a stream status code on the BMB lines defining all bus connections within the stream. Once assigned to a stream a processor always uses the pair of busses specified by the stream status code for communication to and from memory eliminating bus contention among processors of a given type. For redundancy each processor can output on a choice of two busses. This choice is made by BOSS command.

The ARMMS priority structure will involve both hardware and software elements. The hardware recognizes a minimum of 16 different priority levels. The software then selects different subsets of these 16 as program requirements dictate. The highest hardware priority goes to BOSS since the efficiency of the rest of the system depends on BOSS completing its tasks efficiently. The second highest priority is a special TMR CPE mode used only in the event of an error in one of three TMR channels to insure completion of the TMR task with maximum speed prior to initiating diagnostic tests on the stream. The next seven priorities are for I/O streams on the assumption that the timing of external events happening and mass data transfers is more difficult to control than the timing within processing streams and hence IOP memory access requests should be given higher priorities than CPE access requests. The seven lowest priorities are for CPEs.

So long as BOSS, I/O, and CPE programs are mostly segregated into different memory pages all 3 types of programs should be able to be executed simultaneously with minimal bus or memory contention. When these programs wish to access the same memory page the internal logic design of the memory access logic will tend toward letting the streams access the memory a word at a time in turn since each processor will release the memory temporarily between access requests letting the next higher priority stream gain access for one word. This results in all contending streams slowing down but none stopping entirely. Obviously this does not preclude the need for designing the software to minimize memory contention if ARMMS is to perform efficiently as a multiprocessor.

4.2 Memory Module Reliability and Register Level Design Study

It is likely that the least reliable of the ARMMS modules will be the main memories due to the large number of discrete components and small scale integrated circuits required and the power levels associated with accessing the plated wire planes. Fortunately, however, analysis has shown that due to their organization it is possible to achieve 99+% memory reliability on a system basis through judicious use of error detecting and correcting codes which are generated and checked within processor modules and stored in each memory word, internal redundancy within memory modules, spare modules, and duplex memory operation for duplex or TMR processing streams. Software read-after-write in the simplex mode and duplication of data from a good memory into a spare memory in duplex or TMR modes would also be desirable. Using these techniques the results shown in Tables I and II have been obtained. Table I summarizes probabilities of occurrence of dominant failure modes along with recommended solutions while Table II lists various causes of memory failures again with their contributions to the memory module's failure rate. A block diagram of the proposed memory module is shown in Figure 3. The failure rates were derived from data in a 1971 Autonetics Space Station Study. The memory is assumed to use plated wire technology in an 8192 word by 39 bit (32 data bits plus error correcting codes) organization. Some differences will be noted between Table I and II and similar ones in the Phase II report due to the use of updated failure rate data. The rates in the original study were more representative of the late 1960s than of the early 1980s and hence showed the memory in an excessively pessimistic light when compared with other ARMMS modules. Failure rates given for all modules in this report are believed to be consistent.

4.2.1 Memory Module Register Level Design

Plated wire technology was chosen for the ARMMS main memory because of its low, power, weight, and volume and non-volatility in the presence of power transients. Such memories are being used extensively in space computers being designed today for these reasons. The basic organization consists of a 512 word by 628 bit structure which is accessed in a 2-1/2 D configuration requiring 512 word drivers, a 628x39 low level bit multiplexer and 39 bit-switch/sense amplifier circuits allowing 32 data bits plus 7 error detecting/correcting code bits per word. The memories' cycle time is assumed to be 600 nsec for READ and 800 nsec for WRITE. The details of the memories' control and voting logic were discussed in the configuration and error correction sections respectively of the Phase II report. The remaining logic is straightforward except for noting that since the memory must sometimes output data on one bus while the

TABLE I. DOMINANT MEMORY FAILURE MODES AND RECOMMENDED SOLUTIONS

1. Wrong Output of a Single Bit in Each of a Group of Words	
Cond Prob/Given Failure =	~0.600
Solution — All Modes	Hamming-Parity Error Masking Code
2. No Output of all Bits in a Group of Words	
Cond Prob/Given Failure =	~0.220
Solution — Simplex	All "0" Output → Parity Error → Detection
— Duplex	Voter/Switch Output "1" on Disagreement → Masking
— TMR	Majority Vote → Masking
3. Selection of Two Words in Memory at Once	
Cond Prob/Given Failure =	~0.175
Solution — All Modes	Employ Series Redundant Word Drivers to reduce this prob to 0.0014
4. Improper Memory Output Synchronization	
Cond Prob/Given Failure =	0.005
Solution — Simplex	None
— Duplex	Detect Disagreement at Voter/Switch
— TMR	Vote and Mask at Voter/Switch

address for the next cycle is being inputted on another a one word Access-Request Buffer is required to hold the current address stable until the end of the memory cycle.

4.2.2 Memory Reliability Analysis

The dominant failure mode of Table I can be masked by a single error correcting code. The second mode can be detected by such a code if the code bits are inverted prior to storage so that a code check on a word consisting of all "0" will fail. The third mode is the most serious because it can cause properly coded words to be written or read from the wrong location in memory undetected. It is caused by a stuck-on "1" condition in one of the hundreds of plated wire word line drivers. By employing series redundancy in these drivers, the conditional probability of occurrence of this condition can be reduced to a negligible value. Series parallel redundancy (quadding) in these drivers will also virtually eliminate the principal cause of the second failure mode. However, it is probably preferable to provide additional spare memories rather than to resort to quadded word drivers due to the large hardware increase involved in quadded word drivers.

The mean failure rate of a memory employing single error correction coding and serial redundant word drivers is less than half that of a memory

TABLE II. ARMMS MEMORY FAILURE MODES

Component Failing	Result	Failures/10^6 Hours Basic Memory	Failures/10^6 Hours Enhanced Reliability Memory	Corrective Action
Word diodes, switches, mux drivers stuck on "0" – current source or power supply failure	No output (whole words)	0.565	1.105 (0.013)*	Detect with Inv Hamming-Parity code
Word diodes, switches, mux drivers stuck on "1"	Select 2 words at once	0.540	0.013	Not always detectable
Plated wire or sense amp failed; Mux or bit current switch open or short	Single bit failed	1.525	1.790/ 0.072**	Correct with Hamming-parity code
Control logic failures	Select wrong address	0.005	0.005	Detect and inhibit with parity code
	No response to access request	0.010	0.010	Processor timing check
	No output (whole word)	0.060	0.060	Detect with inv Hamming-parity code
	Single bit failure	0.060	0.060/ NIL**	Correct with inv Hamming-parity code
	Detectable garbled output	0.060	0.060	Detect with inv Hamming-parity code
	Parity checker failure	0.005	0.005	Detect with Software
Total Failure Rate		2.83	3.10/ 1.33**	

*Number in ( ) assumes quadded word drivers — not recommended due to excessive hardware involved.
**First number is probability of correctable failure, second number is probability of detectable but not correctable failure.

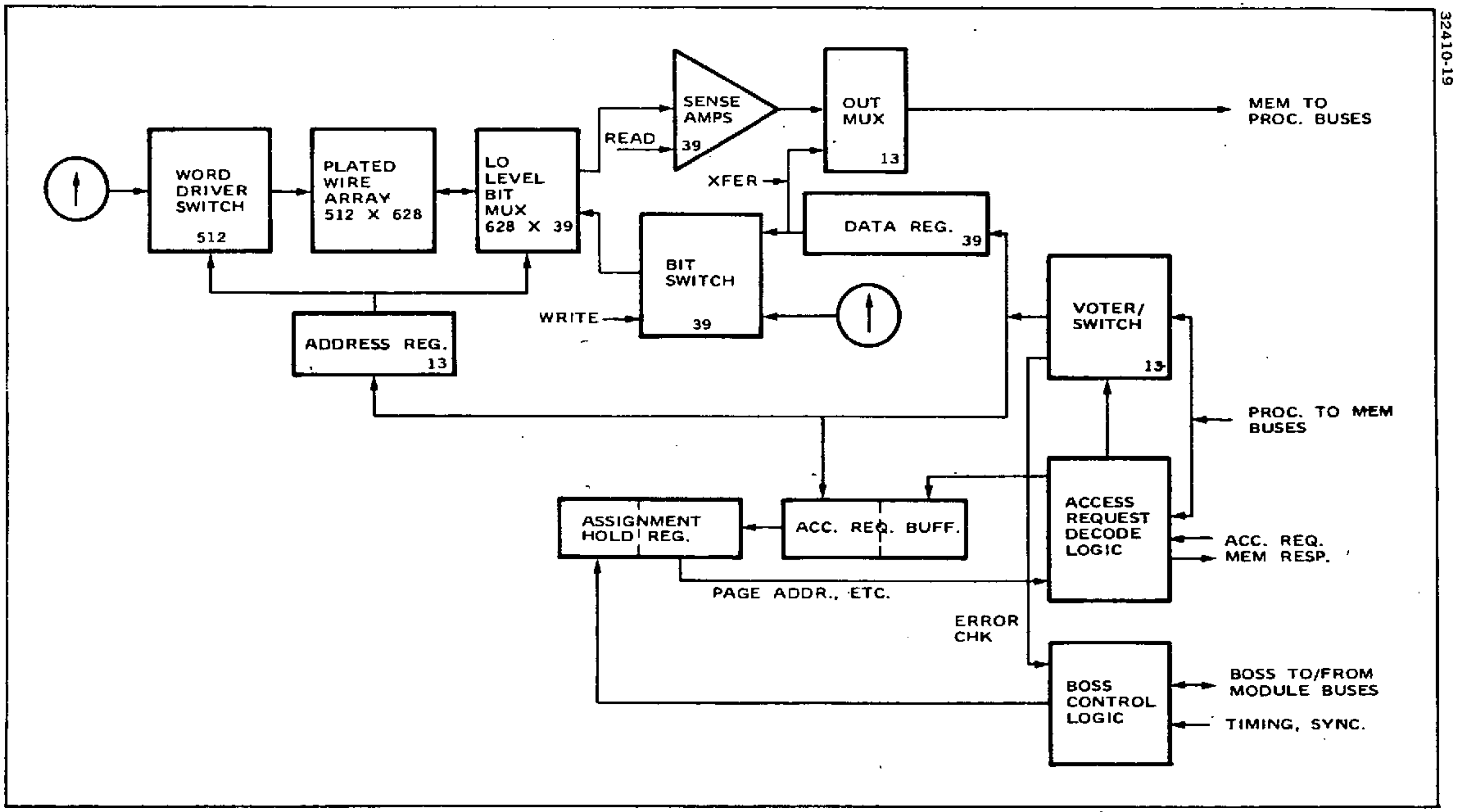


Figure 3. ARMMS Main Memory Functional Block Diagram

without these features. What is more, undetectable failures make up less than 0.5% of the total failure modes yielding a coverage in excess of 0.995. Duplex or TMR operation is required when it is desirable to avoid program rollbacks in the event of a non-correctable memory failure.

In duplex or TMR operation the contents of the good memory can be written into a spare module or used in simplex for the duration of the program. In simplex operation it is essential to avoid writing bad data into the memory, or good data into the wrong location. The former condition can be protected against by immediate verification of all written data by reading out the same location immediately after writing into it in a simplex program. If the data is wrong the procedure can be repeated until the WRITE is accomplished successfully. The latter condition can be protected against by employing an address parity check code at the memory and inhibiting WRITE operations any time address parity is violated. Parity checkers can be provided in each memory at a small hardware cost.

Another result worth noting is that as the number of memory modules required goes up, the ratio of operating modules to required spares decreases, making the use of spares vs internal redundancy more attractive for larger numbers of modules required. A memory module incorporating a single error correcting code and series redundant word drivers should have a probability of surviving a 5-year mission of 0.943 (compared to 0.883 for a memory without these features). This means that if 5 modules are used there will be a 0.9983 probability that at least 3 will be operating after 5 years. If 10 modules are flown there will be a 0.9983 probability that at least 7 survive.

4.3 ARMMS BOSS Register Level Design and Reliability Study

A register level design and reliability analysis were performed for BOSS along with a basic instruction set and list of macro instructions. A partitioned BOSS module should be capable of achieving a reliability of 0.9999 over a five year mission and would require approximately 125 LSICs (of 250 equivalent gate complexity each) to implement.

BOSS will execute routines for data scheduling, system test, repair, and configuration, and interrupt processing. For four simultaneous processing streams executing programs of an average of 5 msec duration BOSS will execute at least 800 routines per second. To meet these function and speed requirements, BOSS will have to be a small special purpose computer including such instructions as LOAD, STORE, NO OP, JUMP, TEST, SPCJ, AND, OR, SHIFT, ADD, SUB, plus macro instructions to speed up frequently used processes such as table searches requiring correlations and list processing.

BOSS will look functionally similar to the SUMC CPE – however SUMC instructions such as Multiply, Divide, Square Root, Floating Point and double precision will not be needed and special system monitoring and control logic will be required in BOSS but not in the CPE. BOSS will be capable of accessing and testing half-words, bytes, bits, multiple words and variable length fields for efficiency in list handling. If a modified SUMC related design is used for BOSS, speed requirements would limit average BOSS program lengths to about

875 operations per task assuming 4 streams operating simultaneously with a 5 msec average task length. Individual BOSS processor partition complexity is expected to be 60% of the present SUMC complexity or 90% of that of an ARMMS CPE based on a modified SUMC processor.

Originally BOSS was envisioned as a group of identical modules any three of which could be operated in TMR to provide ultra-high reliability. However, BOSS will have nearly 300 system level interconnects and if a group of BOSS processors were used each one would need almost this many interconnects. In addition, with individual BOSS processor modules, location of BOSS power and configuration control, command voting, oscillator and power supply logic becomes a problem. One solution is to group these functions into a very simple and hence very reliable internally redundant "super-BOSS" module. The interconnect problem which can effect both volume and reliability is solved by grouping the BOSS processors and the "super-BOSS" physically into one module requiring only one set of system level interconnects. The BOSS processors and the "super-BOSS" become partitions "A" and "B" respectively. Reliability estimates based upon BOSS register level design indicates that 4 "A" partitions and 2 "B" partitions should meet ARMMS reliability goals.

4.3.1 BOSS Reliability Analysis

By operating BOSS in at least a duplex mode (and in TMR so long as possible) failures in most BOSS logic will be detected – including those in the ALU and control logic blocks. Parity checks can be performed inexpensively on BOSS memory and the Hamming Parity logic is required in order to check the main memory, keeping the cost associated with self-checking BOSS to a minimum. If BOSS detects and isolates an error to a memory module it is accessing it generates an internal interrupt allowing the executive software's memory replacement routine to be actuated.

BOSS is estimated to have a 0.9999 probability of successful duplex operation after 5 years and a 0.9957 probability of continued TMR operation over that period, assuming 4 partitions "A" are flown. These reliability figures assume the register level designs shown in Figure 4. Table III lists BOSS failure modes along with resultant error patterns, failure rates and suggested corrective action as a function of the component block failing. The CPE Register Level design and Reliability Study topic in this report includes CMOS LSIC functional partitioning estimates for both BOSS and CPE modules. It is expected that most BOSS integrated circuit designs would be usable in the CPE as well. BOSS partition "A"s are anticipated to use 31 chips of 10 different types. The dashed lines in Figure 4 delineate these partitions.

Referring to Figure 4, similarities can be seen between BOSS and SUMC since SUMC was used as a starting point. However, the memory Input and Instruction registers are duplicated to allow for instruction overlapping, there is no MQ register or floating point unit since these functions are not needed in BOSS, error detection logic and bus interfaces and voting logic have been added along with priority interrupt and interval timer logic and the ALU-multiplexer structure has been simplified. At a detailed level radical changes are expected in the structure of the microprogram read-only-memory and scratchpad memory and in general the design has been simplified and streamlined to increase the processor's speed and ease error detection and correction. Hence SUMC LSI

TABLE III. BOSS FAILURE MODES

Components Failing	Result	Failure/ 10^6Hr	Corrective Action
Input mux or voter/switch	Triple-bit error	0.043	Detect with H-P code-inhibit out
Mem in, addr, data reg output and ALU muxes	Single bit error	0.125	Detect and mask with H-P code
Scratchpad memory (SPM), MQR	Single bit error	0.130	Detect with parity check, inhibit out-est coverage = 0.95
Arithmetic logic unit	Multiple-bit error	0.102	Detect in duplex, mask in TMR-est coverage = 0*
Microprogram ROM (MROM)	Control bit error	0.125	Detect with parity check, inhibit out-est coverage = 0.9
Instruc reg and mux	SPM or MROM addr bit error	0.031	Detect by comparing with memory in reg-inhibit out
Interrupt reg, iter ctr, seq and mem acc contr, clock regs	Improper execution, loss of sync	0.086	Detect in duplex, mask in TMR-est coverage = 0
Error detection logic	False error indication	0.108	Inhibit Output
Total		0.750	

*This failure mode could be detected and masked internal to the partition by duplication of the ALU and comparing outputs. Since BOSS is not to be operated in simplex this redundancy is not necessary nor recommended although it is desirable in processor modules.

modules are not likely to be useful for BOSS and the CPE should probably be an extension of the BOSS design rather than a modification of SUMC in order to maximize commonality and minimize cost within ARMMS.

Referring to Table III it can be seen, assuming no duplication of the ALU logic within a BOSS partition and at least duplex operation, that nearly 75% of BOSS failure modes will be maskable and that virtually all will be detectable. In TMR operation, virtually all failures can be masked. The numbers in the table are also representative of CPE failure rates except that with duplication of ALU and floating point arithmetic logic the conditional probability of being able to detect a failure given that one occurs while operating with simplex mode rises accordingly. As noted earlier simplex operation of BOSS is not necessary or desirable in ARMMS while simplex processor operation is both to be expected

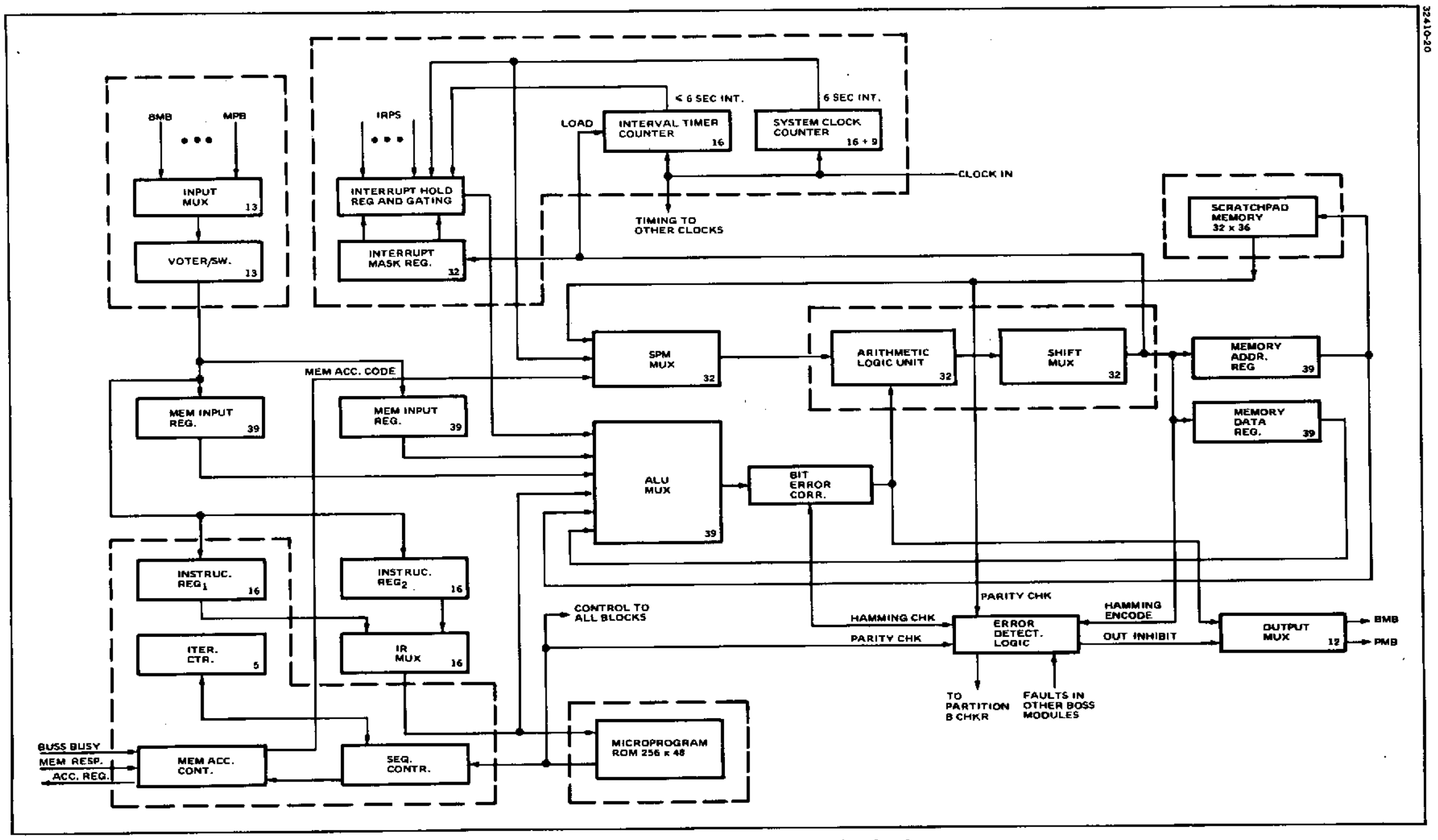


Figure 4.. ARMMS BOSS Functional Block Diagram

and desirable. Note that parity checks are made on BOSS internal memories. Certain on-chip addressing logic problems are not detectable with a parity check therefore memory coverage is expected to be 0.9 to 0.95 rather than unity. Coverage is assumed to be unity in the tables "corrective action" column except as noted. When one partition's output is inhibited, the memory module's voter/ switch will mask this output allowing the other partition's correct output to propagate to the memory. The same thing is true of partition "B" command voting logic.

Given a non-maskable failure in a BOSS partition the replacement algorithm implemented a partition "B" is as follows:

1. Power on partitions 1, 2, 3 at the start of the mission.
2. Replace the first failing partition with partition 4 (prob 0.1040).
3. Power off the second failing partition – BOSS is now in duplex operation (prob 0.0043).
4. If a third, non-maskable failure occurs (prob 0.0001) ARMMS will cease operating and wait for outside assistance. Retrying BOSS partitions can be done on command but will not be done automatically since this won't necessarily correct the failure and can lead to undetected erroneous computations being outputted from the computer.

Partition B is statistically very reliable but conservative design calls for providing a spare partition to be switched in automatically upon self-detected disagreement within the first partition.

4.3.2 BOSS Register Level Design

The BOSS microprogram read only memory organization is summarized in Figure 5. Bits have been provided to implement all BOSS micro and macro instructions discussed later in this section. This MROM would have to be modified for CPE operation. Fields are included for interrupt, interval timer scratchpad memory, ALU, Multiplexer, hardware register, bus interface, and sequencer control functions. Each MROM word requires 42 bits plus parity and 256 words are provided reducing the memory to 15% of the size of the one in SUMC.

Figure 6 shows the BOSS instruction and data formats. Three types of instructions and one data format are recognized: Main memory reference instructions contain an address for a 2nd operand fetch including a choice of 3 index registers, 4 base registers plus a no index register option when its field is "0". An 8 bit op-code accesses the MROM directly with the op-code of an instruction being the MROM address of its first micro instruction. Two register addresses are provided for accessing two words from scratchpad memory during the course of the instruction. Field R_1 can access any non-privileged SPM location while Field R_2 accesses 8 of the accumulators. Single operand instructions have formats the same as above except that a third general SPM register may be accessed with Field R_3 rather than a main memory location. Link word

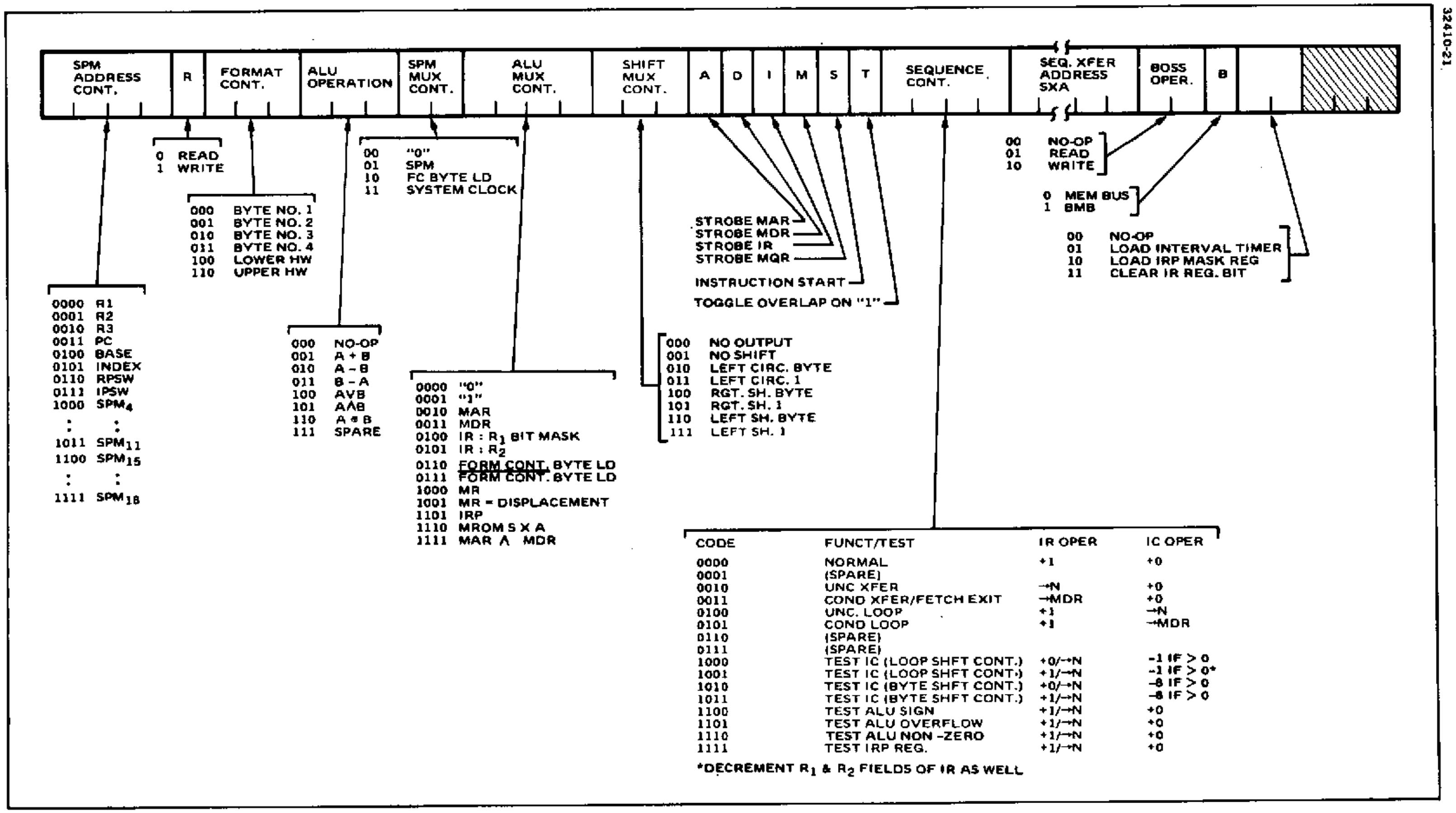


Figure 5. BOSS Microprogram Memory Organization

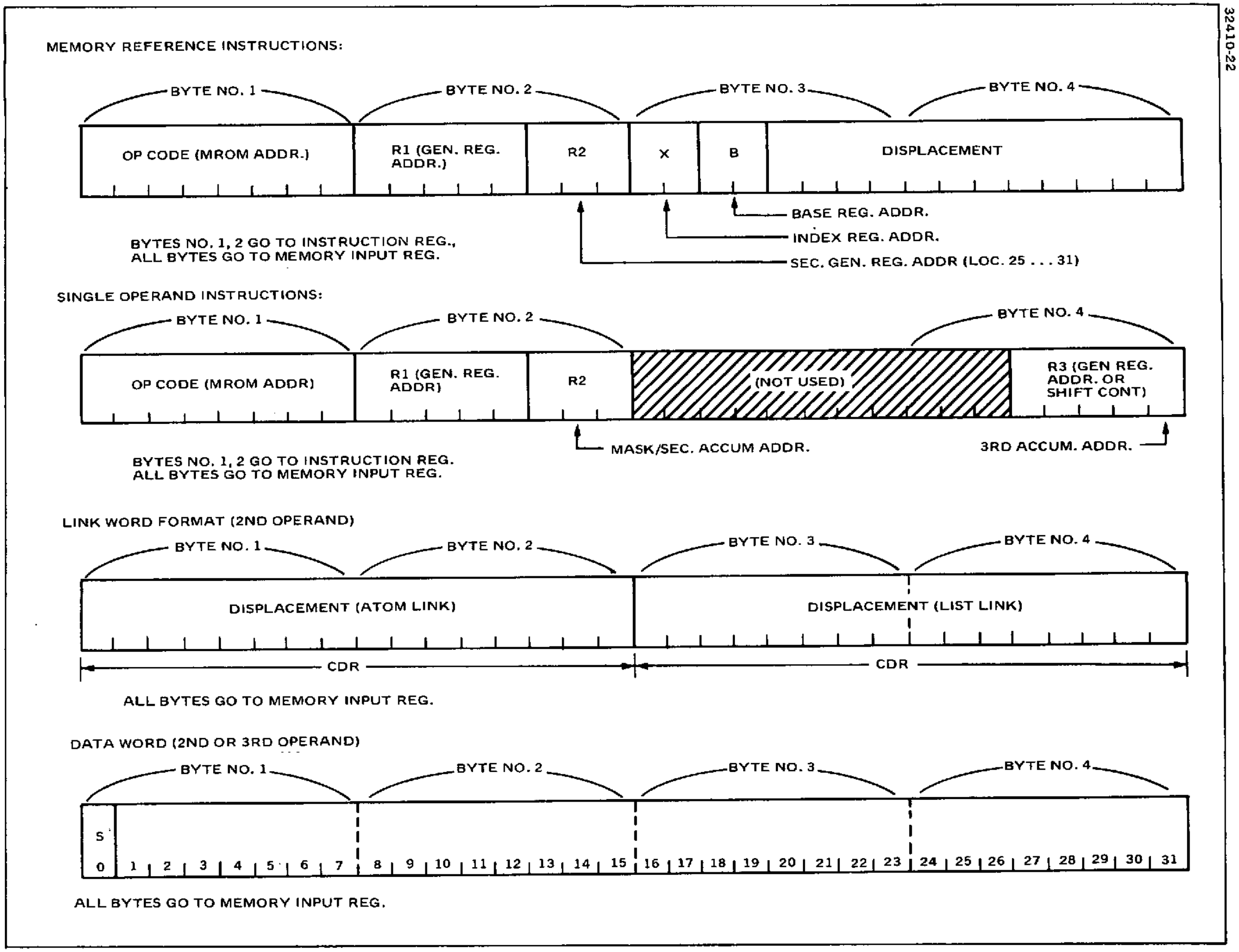

32410-22
MEMORY REFERENCE INSTRUCTIONS:
BYTE NO. 1
BYTE NO. 2
BYTE NO. 3
BYTE NO. 4
OP CODE (MROM ADDR.)
R1 (GEN. REG. ADDR.)
R2
X
B
DISPLACEMENT
BASE REG. ADDR.
INDEX REG. ADDR.
SEC. GEN. REG. ADDR (LOC. 25 . . . 31)
BYTES NO. 1, 2 GO TO INSTRUCTION REG., ALL BYTES GO TO MEMORY INPUT REG.
SINGLE OPERAND INSTRUCTIONS:
OP CODE (MROM ADDR)
R1 (GEN. REG. ADDR)
R2
(NOT USED)
R3 (GEN REG. ADDR. OR SHIFT CONT)
MASK/SEC. ACCUM ADDR.
3RD ACCUM. ADDR.
BYTES NO. 1, 2 GO TO INSTRUCTION REG. ALL BYTES GO TO MEMORY INPUT REG.
LINK WORD FORMAT (2ND OPERAND)
DISPLACEMENT (ATOM LINK)
DISPLACEMENT (LIST LINK)
CDR
CDR
ALL BYTES GO TO MEMORY INPUT REG.
DATA WORD (2ND OR 3RD OPERAND)
S
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
ALL BYTES GO TO MEMORY INPUT REG.

Figure 6. BOSS Instruction and Data Formats

instructions are used for list handling and provide two main memory address fields allowing indirect address linkage to a data item in main memory and to the next link in the list. Data words allow 32 bit signed fixed point data to be accessed by BOSS.

Figure 7 shows the organization of the BOSS scratchpad memory. It contains 21 accumulators, plus 3 index registers directly accessible by the program. In addition a rollback program status word (RPSW) and interrupt status word (IPSW) provide for program jumps on errors and interrupts and five base registers provide for extended main memory access when summed with an instruction displacement field. The RPSW, IPSW, and Program counter are read accessible but not write accessible under normal conditions. The Index Registers can be used as additional accumulators. Seven of the accumulators can be specified by the MROM for use as working storage during macroinstruction execution. The base registers not accessible by the instruction's base field are accessed automatically during program and subroutine branch instructions.

A system clock counter and a separate interval timer counter are included in the BOSS module. The system clock counts for a 6 second interval with 100 μsec resolution. Longer periods are stored in a software counter in BOSS's portion of main memory in response to a system clock interrupt. The interval timer provides an interrupt at the end of a software specified interval of up to 6 seconds with 100 μsec resolution.

BOSS includes a priority interrupt structure in which up to 32 priority levels may be provided by software specification of a hardware interrupt masking register's contents. Only interrupts corresponding to "1" bits in this register will be responded to and cleared in the interrupt holding register allowing the software to establish varying priorities for different interrupts and to complete processing a given set of interrupts without further interruption from interrupts of lower priority. The assumed hardware and firmware roles in the interrupt structure are shown in Figure 8.

4.3.3 BOSS Interaction with Other Modules

BOSS will command and interrogate other modules via a 2-way BOSS/Module bus (BMB). Each module will contain bus interface logic capable of decoding a unique access code for that module plus a general sync code which allows simultaneously starting or stopping several pre-primed processors working together in the same stream. The interface logic will also gate the module's status word MSW onto the BMB in response to a transmit MSW command from BOSS to the module. Both processor and memory MSWs would contain their BOSS assignments (memory page, processor bus access priority and stream assignment codes) and in addition memories could use a one bit code to indicate failures and the CPEs would include a 7 bit status code including a 2 bit hardware determined error code and a 5 bit software determined termination code.

BOSS would then use the code to determine which subroutine to branch to in response to the processors' status. BOSS could interrogate processors periodically or in response to interrupts from them. Descriptions of, and formats for, BOSS commands to other modules are shown in Figure 9. The "save" and "restore" data commands cause the processor to store or load data respectively from an area of memory defined in the commands. This allows BOSS

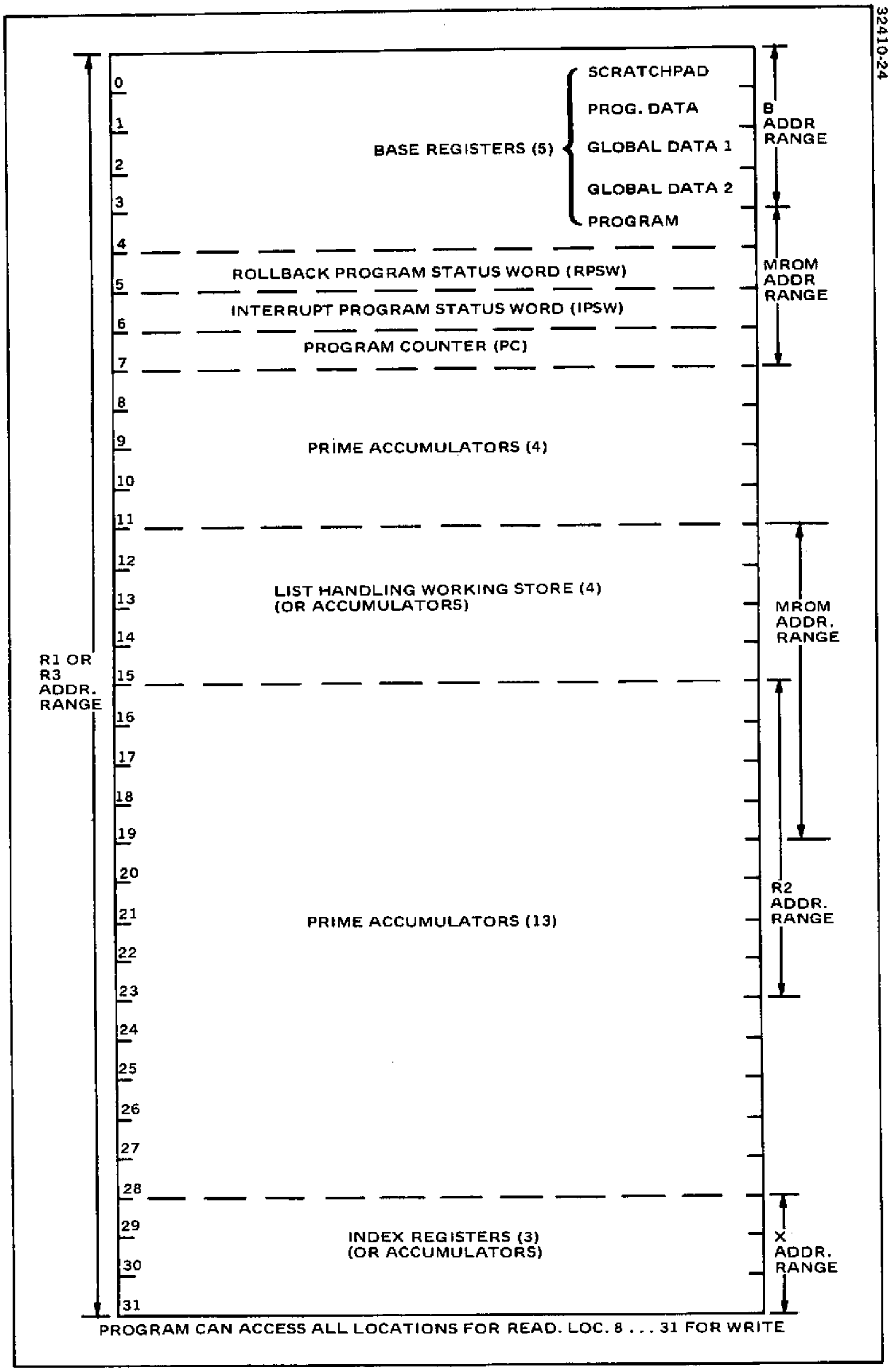
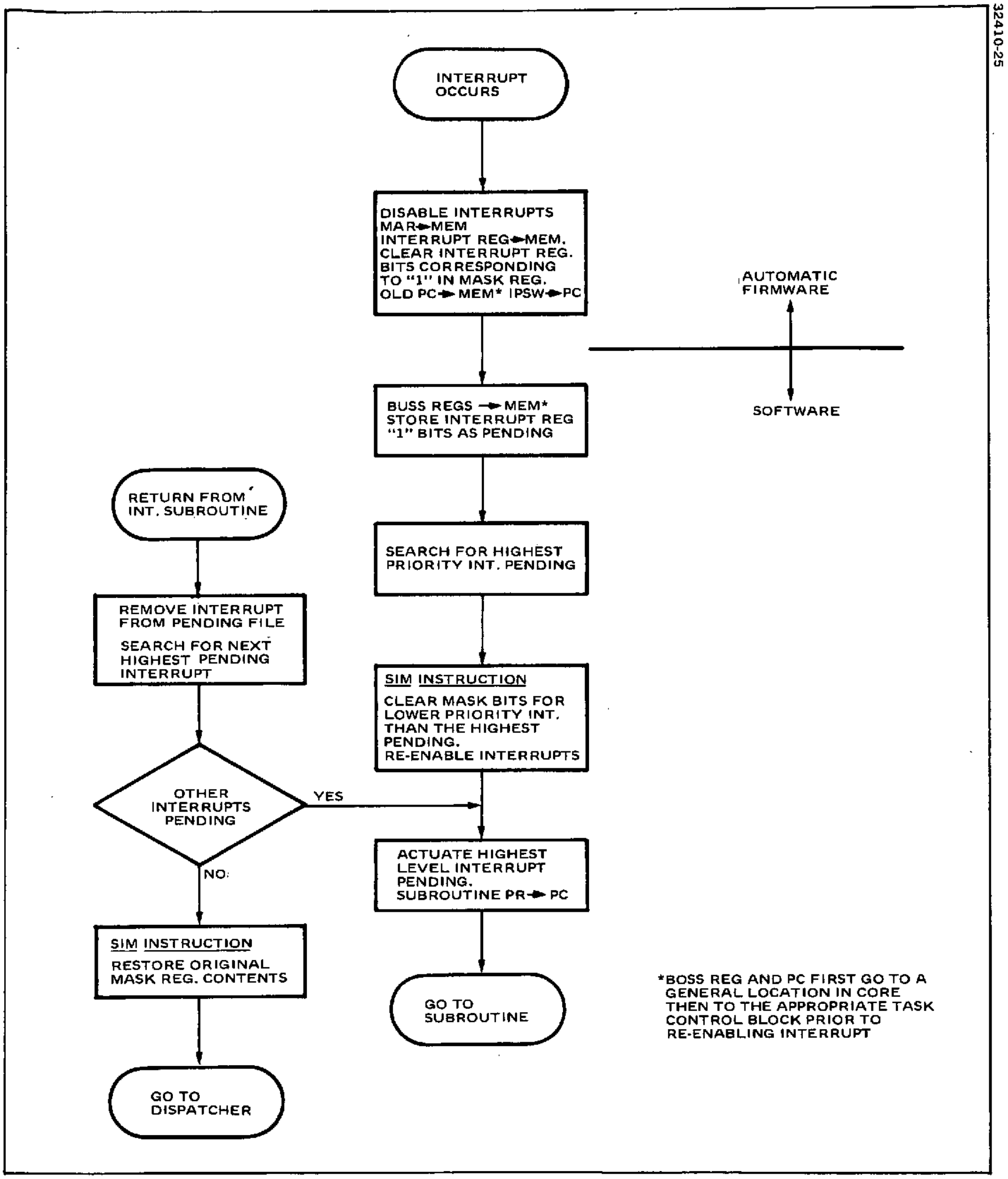


Figure 7. BOSS Scratchpad Memory Organization



32410-25
INTERRUPT OCCURS
DISABLE INTERRUPTS
MAR→MEM
INTERRUPT REG→MEM.
CLEAR INTERRUPT REG.
BITS CORRESPONDING TO "1" IN MASK REG.
OLD PC→MEM* IPSW→PC
AUTOMATIC FIRMWARE
SOFTWARE
BUSS REGS →MEM*
STORE INTERRUPT REG "1" BITS AS PENDING
SEARCH FOR HIGHEST PRIORITY INT. PENDING
SIM INSTRUCTION
CLEAR MASK BITS FOR LOWER PRIORITY INT. THAN THE HIGHEST PENDING.
RE-ENABLE INTERRUPTS
RETURN FROM INT. SUBROUTINE
REMOVE INTERRUPT FROM PENDING FILE
SEARCH FOR NEXT HIGHEST PENDING INTERRUPT
OTHER INTERRUPTS PENDING
YES
NO
ACTUATE HIGHEST LEVEL INTERRUPT PENDING.
SUBROUTINE PR→PC
SIM INSTRUCTION
RESTORE ORIGINAL MASK REG. CONTENTS
GO TO SUBROUTINE
GO TO DISPATCHER
*BOSS REG AND PC FIRST GO TO A GENERAL LOCATION IN CORE THEN TO THE APPROPRIATE TASK CONTROL BLOCK PRIOR TO RE-ENABLING INTERRUPT

Figure 8. BOSS Software/Firmware Interrupt Handling

BOSS TO MODULE COMMANDS

CODE	COMMAND	MEMORY	PROCESSORS	ARGUMENT (6)
00	STOP – SAVE DATA ** – PRIME FOR SYNC STOP		X	MEMORY ADDR.
01	RESTORE DATA* - PRIME FOR SYNC START		X	MEMORY ADDR.
10	TRANSMIT MSW	X	X	SUBCODE = 0
11	LOAD ASSIGNMENT REG	X	X	ASSIGNMENT
10	SYNC START		X	SUBCODE = 1

*GIVES BOSS WRITE ACCESS TO PRIVELEGED BASE/BOUND REGISTERS.
**CAUSES AUTOMATIC CPE SCRATCHPAD MEMORY DUMP

FORMAT:

	SYNC	PARITY	DATA	
TIME t	1	P	8 BIT 2 of 4 CODED ADDRESS	
TIME t + 1	0	P	CODE (2)	ARGUMENT OR SUBCODE (6)

FIGURE 9

access to the processor's registers including privileged Base/Bound registers not accessible by general programs. Transmission on the BMB will be parity coded and a sync line is included to activate modules' access decoders. The BMB is duplicated so that modules can verify accuracy of commands through comparison of signals on the 2 buses and BOSS can likewise verify data from the modules.

4.3.4 BOSS Instruction Set

BOSS microprogrammed firmware includes 29 general purpose instructions plus 37 specialized macroinstructions. BOSS macroinstructions cover bit and byte testing, byte, half-word, multiple word, and field load and store instructions, a set of instructions for formation and manipulation of linked lists, and instructions for interrupt handling and communication with other ARMMS modules. These instructions were designed to allow rapid, efficient manipulation of various tables, lists, queues, and other data structures contained in BOSS memory. The macroinstructions, as listed in Table IV, use an estimated 115 words of microprogram read-only-memory, and have an average execution time of 1.7 μsec each, assuming 10 MHz system clock. This compares with 29 basic instructions listed in Table V having an average execution time of 1.4 μsec and requiring 93 words of microprogram storage.

4.4 ARMMS CPE Register Level Design and Reliability Study

A register level design and reliability analysis were performed for the ARMMS CPE module along with a study of CPE/BOSS/IOP commonality. The CPE is based on a SUMC design extensively modified for increased performance reliability and compatibility with ARMMS system requirements. The ARMMS CPE requires 35 LSICs (of 250 equivalent gate complexity each) and should exhibit a failure rate of $\approx 0.85 \times 10^{-6}$ failures/hour and have 80% commonality with BOSS partition "A" and IOP logic. The CPE requires 1.2, 5.0, and 9.6 μsec to perform addition, multiplication, and floating point division instructions respectively assuming a 5 MHz clock and overlapped instruction fetching as in the case of BOSS.

4.4.1 CPE Commonality with BOSS and IOP

As noted earlier in this report, making the BOSS and CPE modules identical does not appear to be desirable. However, accomplishing identical functions within both CPE, IOP, and BOSS modules with identical LSI chip designs does appear feasible and should minimize system development costs since fewer different chip types need to be developed and tested. With this in mind an LSI partitioning study was conducted and 18 LSI chip types were tentatively identified and are listed in Table VI. Of these 18, 8 are used both in BOSS IOP and CPE modules, 1 is used exclusively in the CPE, 1 in the CPE and IOP, 2 are used exclusively in BOSS and 6 are used exclusively in the IOP. In terms of total chip quantities the modules each have 28 chips in common out of a total of 36 for the CPE, 45 for the IOP and 32 for BOSS.

In the partitioning study the number of gates ranged from 180 to 270 per device while the number of pins ranged from 20 to 80 per device. The assumed number of gates is realistic in terms of near future CMOS silicon-on-sapphire technology as are 5 MHz clock rates and 0.5 watt/chip power dissipations but

TABLE IV. BOSS MACRO INSTRUCTIONS DESCRIPTIONS

Mnemonic	Instruction	Timing Cycles	Microprogram Storage
LIT	Load and start interval timer	8	1
RSC	Read system clock reg	8	1
SIM	Set interrupt mask	8	1
LRR	Load rollback reg from prog ctr	21	2
COM	Command module via BMB	12	1
INM	Interrogate module via BMB	12	1
XCR	Exchange registers R_1 and R_2	16	5
XFR	Transfer reg R_1 to reg R_2	10	2
LMR	Load multiple registers	14 + 6n*	4
SMR	Store multiple registers	14 + 9n*	4

*These instructions allow loading or storing of n = 1 ... 8 contiguous registers as specified by R_2, starting at locations R_1 in scratchpad memory and A in main memory.

Generalized "Clear and Add" and Store Instructions

Mnemonic	Function
LB1, SB1	Load, store byte 1
LB2, SB2	Load, store byte 2
LB3, SB3	Load, store byte 3
LB4, SB4	Load, store byte 4
LH1, SH1	Load, store half-word 1
LH2, SH2	Load, store half-word 2
LDA	Load address
LIH1, SIH1	Load, store indirect half-word 1
LIH2, SIH2	Load, store indirect half word 2

	Timing	Microprogram Storage
Direct loads	1.2 μsec	1 word
Direct stores	1.5 μsec	1 word
Indirect loads	1.8 μsec	2 words
Indirect stores	2.1 μsec	2 words

All byte and half word instructions right-justify bits on loads and assume right-justified bits on stores.

TABLE IV. BOSS MACRO INSTRUCTIONS DESCRIPTIONS (Continued)

Generalized Test Instructions (Arguments are assumed to be stored in respective registers prior to execution of these instructions)

BON	R_1, R_2, A	Branch if bit on
BOF	R_1, R_2, A	Branch if bit off
TUM	R_1, R_2, A	Test under mask, branch on equal
TDM	R_1, R_2, A	Test under mask, branch on equal, else decrement index

R_1	=	Bit No. to be tested in BON, BOF
R_1	=	Genl reg to be compared with memory in TUM, TDM
R_2	=	Branch distance in all instructions
A	=	Address (incl base and index) of memory location under test
$R_2 + 1$	=	Address of 32-bit mask in TUM, TDM

Index register to be decremented in TDM is specified by the X portion of A.

Timing:	BON, BOF = 1.5 μsec	TUM = 1.7 μsec	TDM = 1.9 μsec
	BON, BOF = 3	TUM = 4	TDM = 5 words

Generalized Partial Word Instructions (Arguments are assumed to be stored in respective registers prior to execution of these instructions)

CLF,	R_1, R_2, A	Clear and add masked field
STF	R_1, R_2, A	Store masked field

R_1 = Genl reg to be loaded or stored from

R_2 = Addr of 32-bit mask — bits of R or A corresponding to

Mask postions containing "1" will be changed, remaining bits will not be changed.

A = Address (incl base and index) of memory location containing bits in question.

Timing = CLF = 1.2 μsec	STF = 2.7 μsec
Memory Est: CLF = 1	STF = 7 words

TABLE IV. BOSS MACRO INSTRUCTIONS DESCRIPTIONS (Continued)

List Manipulation Instructions – (arguments are assumed to be stored in respective registers prior to execution of these instructions).

BOSS Specification	Function
NXT R_1, - , - ,	Step to next item
INS R_1, - , A	Insert A after W
RMV R_1, - , -	Remove W
FND R_1, R_2, R_3	Find item according to mask

R_1	Word offset in the (assumed) atom to be fetched
R_2	Mask with "1" bits in bit positions to be compared
R_3	Genl reg containing word for comparison
A	Pointer address

Timing: NXT = 2.6, INS = 5.4, RMV = 4.0, FND = 1.8 + 3.6 μsec/item

Memory Est: NXT = 8 INS = 15 RMV = 11 FND = 15 words

TABLE V. TENTATIVE BASIC BOSS INSTRUCTION SET

Mnemonic	Instruction	Avail. in SUMC	Timing μsec	Microprogram Storage
JRE	Jump On Register Equal to Memory	Y	1.5	4
JRG	Jump on Register Greater than Memory	Y	1.4	2
JRN	Jump on Register Not Equal to Memory	N	1.5	4
JRL	Jump on Register Less than Memory	N	1.4	2
SPJ	Store Program Counter and Jump	N	2.1	2
JMP	Jump Unconditionally	Y	1.4	1
JPI	Jump Unconditionally Immediate	Y	1.4	1

TABLE V. TENTATIVE BASIC BOSS INSTRUCTION SET (Continued)

Mnemonic	Instruction	Avail. in SUMC	Timing sec	Microprogram Storage
XEC	Execute	N	0.8	1
ADM	ADD Memory to Register	Y	1.2	2
SBM	Subtract Memory from Register	Y	1.2	2
ANM	AND Memory with Register	Y	1.4	2
ORM	OR Memory with Register	Y	1.4	2.
XOM	Exclusive OR Memory with Register	Y	1.4	2
ADR	ADD Register to Register	Y	1.2	4
SBR	Subtract Register from Register	Y	1.2	4
ANR	AND Register with Register	Y	1.2	4
ORR	OR Register with Register	Y	1.2	4
XOR	Exclusive or Register with Register	Y	1.2	3
ICT	Increment Memory	N	2.3	3
NOT	Complement Register	N	1.6	3
DLY	Delay N Cycles	Y	0.8	1
HLT	Halt and Wait for Interrupt	Y	0.8	1
CWM	Compare Register with Memory	N	2.2	4
CSR	Compare Register Selectivity with Register	N	2.2	5
SHR	Shift Right N Bits	Y	2.0	6
CYL	Cycle Left N Bits	Y	1.8	5
SHL	Shift Left N Bits	N	2.0	5
CLA	Clear and Add Memory	Y	1.2	1
STO	Store in Memory	Y	1.2	1

NOTES: 1. Speeds assume 10 MHz system clocks.
2. Microprogram storage estimates assume an additional 6 word fetch routine.

TABLE VI. BOSS/CPE/IOP LSI PARTITIONING COMMONALITY

Type	Function	Bit Width	Usage BOSS	Usage CPE	Usage IOP	Estimated Pins	Estimated Gates	Coverage
1.	Sequence, memory access, and ALU mux control	-	1	1	1	75	220	0
2.	M, D, sqrt control, BOSS status control and SPM addr control	-	0	1	1	70	180	0
3,4	Hamming-parity error checker	9-12	4	4	4	50	270	1.0
5.	EALU	7	0	2	0	60	230	1.0
6.	ALU	8	4	8	7	30	255	0/0/1.0
7.	Voter switch/in/out mux	5	3	3	3	80	215	1.0
8.	Mux-Register	2-5	8	8	8	70	245	1.0
9.	SPM	9	4	4	6	25	288 Bit	0.95
10.	MROM	10	5	5	5	20	2560 Bit	0.9
11.	Interrupt holding and masking and interval timer	16	2	0	0	45	225	0
12.	System clock and SPM addr mux contr	16	1	0	0	65	185	0
13.	Channel registers	3-16	0	0	2	76	265	0
14.	SPM, channel-mem registers	8-10	0	0	4	50	115	0
15.	Chan-mem interface control	-	0	0	1	60	250	0
16,17	Channel command control, I and II	-	0	0	2	60	250	0
18.	Device interface control	-	0	0	1	60	250	0
			32	36	45			

the number of pins is somewhat optimistic, especially for beam leaded devices. However, the pin requirements will probably not be unrealistic by 1980 if advances in the state of the art continue at their present rate. More refined LSI partitioning studies based on CPE detailed design have been included as part of an ARMS breadboard follow-on to this contract.

CPE Reliability Analysis

A CPE reliability analysis was performed. A summary of potential CPE failure modes indicating component failing, failure rates, and corrective action taken by the CPE as a function of component block failing is listed in Table VII. Internal to each CPE, arithmetic and some control logic is duplicated with outputs compared and parity checks are made on both microprogram and scratchpad memories. In general the reliability discussion of Table III is the BOSS module description also applies to the CPE.

For the CPE module, failure analysis leads to the following additional results:

1. If a CPE is replaced at the end of a task in which it fails, and software is capable of switching the CPE output to a redundant output port if the primary port fails, and redundant ALUs and EALUs are used, the CPE has the following reliability characteristics:

Logic Failure rate/10^6 hours	$\simeq$0.85
Simplex mode coverage	$\simeq$93%
Duplex or TMR coverage	$\simeq$100%
Failures Maskable in Simplex	$\simeq$14%
Failures Maskable in Duplex	$\simeq$93%
Failures Maskable in TMR	$\simeq$100%

2. Power supplies and buss interface electronics failure rates are less than 10% of the logic failure rate.

3. Approximately 33% of CPE logic is devoted to failure detection and correction in the baseline CPE design. This logic detects most memory module failures in addition to those within the CPE. If the EALU and ALU were non-redundant only 20% of CPE logic would be devoted to failure detection and correction and CPE module complexity would be reduced by 15%. However, simplex mode coverage would fall to 76%.

4. Assuming 5 CPES are initially flown, the probability of different numbers of CPE's remaining operational within a 5 year mission is shown below both with and without arithmetic logic redundancy.

No. CPE Operational	With Redundant Arithmetic Logic	Without Redundant Arithmetic Logic
5	0.8300	0.8542
$\geq$4	0.9876	0.9909
$\leq$3	0.9996	0.9997
$\geq$2	0.99999	0.99999

TABLE VII. CPE FAILURE MODES

Components Failing	Result	Failure/ 10^6 Hr	Corrective Action
Input mux or voter/ switch	Triple-bit error	0.043	Detect with H-P code-inhibit output
Mem in, addr, data reg, instr reg, output and ALU muxes	Single bit error	0.125	Detect and mask with H-P code
Scratchpad memory (SPM), MQR	Single bit error	0.130	Detect with parity check-inhibit output, est coverage 0.95
Arithmetic logic unit	Multiple bit error	0.204	Detect by comparison of redundant ALU outputs-inhibit output
Exponent arithmetic unit	Multiple bit error	0.044	Detect by comparison of redundant EALU outputs inhibit output
Microprogram ROM (MROM)	Control bit error	0.125	Detect with parity check-inhibit output, est coverage 0.9
Instruc reg and mux	SPM or MROM addr bit error	0.031	Detect by comparison with mem input reg — inhibit output
Iter ctr, seq and mem access contr, BOSS status and contr interface	Improper execution loss of sync	0.040	Detect in duplex, mask in TMR simplex coverage = 0
Error detection logic	False error indication	0.108	Inhibit output
		0.850	

NOTE: Coverage = 1.0 unless otherwise noted.

This means that only one or two spare CPEs need be flown over and above the number required for use during the mission but that at least one CPE failure may occur and ARMMS should be able to accept it gracefully.

5. Duplication of CPE arithmetic logic seems justified in order to increase simplex error detection coverage if a significant amount of simplex operation is contemplated. Simplex coverage could be increased further by adding additional control unit redundancy but this is probably not worth the effort so long as a duplex mode is available. Increasing simplex error detection coverage also increases duplex error masking.

6. Many missions could be well served without a TMR mode if 93% processor error masking were acceptable rather than 100%.

7. When an error is detected the processor masks the error if possible or else attempts a program rollback. If masking or rollback is successful the processor will not interrupt BOSS for fault correction assistance until its task is completed.

4.4.2 CPE Register Level Design

The CPE register level design is shown in Figure 10. This design was assumed in the reliability discussions above. The dashed lines in the figure represent LSI partitions of Table VI. Referring to Figure 10, similarities can be seen between CPE and SUMC since SUMC was used as a starting point. However, the memory Input and Instruction registers are duplicated to allow for instruction overlapping; error detection logic, bus interfaces, voting logic and BOSS status and control interfaces have been added, and the ALU-multiplexer structure has been simplified. At a detailed level radical changes are expected in the structure of the microprogram read-only-memory and scratchpad memory and in general the design has been simplified and streamlined to increase the processor's speed and ease error detection and correction.

The CPE microprogram read-only-memory organization is summarized in Figure 11. Bits have been provided to implement all CPE micro and macro instructions mentioned earlier in this section. Fields are included for multiplexer, scratchpad memory, ALU, EALU, hardware register, bus interface, and sequencer control functions. This MROM differs from that of BOSS principally in that the interrupt and timer control functions of the BOSS are not required and MQR and Exponent Unit control functions are added. Each MROM word requires 46 bits plus parity and 256 words are provided reducing the memory to 15% of the size of the one in SUMC and slightly larger than the one in BOSS which was 42 bits wide.

Figure 12 shows the CPE instruction and data formats. These are the same as for the BOSS module except that in the CPE data words allow 32 bit signed fixed point data, floating point data including 24 bits plus sign for mantissa and 6 bits plus sign for exponent fields, and double precision fixed and floating point data to be accessed by the CPE. The CPE instruction set shown in

Table VIII is close to that of BOSS with added arithmetic, floating point and double precision instructions similar to those defined in MSFC document S&E - ASTR-004 and the ARMMS Phase II report plus instructions for communication with BOSS.

Figure 13 shows the organization of the CPE scratchpad memory. It contains 14 accumulators, plus 6 base, 6 bound and 3 index registers directly accessible by the program. In addition a rollback program status word (RPSW) and interrupt status word (IPSW) provide for program jumps on errors and interrupts and six base registers provide for extended main memory access when summed with an instruction displacement field. The RPSW, IPSW, and Program counter are read accessible but not write accessible under normal conditions. The index registers can be used as additional accumulators. Seven of the accumulators can be specified by the MROM for use as working storage during macroinstruction execution. The 2 sets of base/bound registers not controlled by the base field of the instructions are used for testing instruction fetches during program and subroutine branch instructions. The organization is similar to that of BOSS except for the base and bounds registers.

4.4.3 CPE Interaction with BOSS

BOSS will command and interrogate other modules via a 2-way BOSS/Module bus (BMB) as discussed in the BOSS description. Each CPE module will contain bus interface logic as shown in Figure 14 allowing it to communicate with BOSS. CPE module status words MSWs contain their BOSS assigned priority and stream assignment codes in addition to a 7 bit status code which includes the 2 bit hardware determined error code shown below and a 5 bit software determined termination code derived from the processor's HALT instruction's R3 field. The options for this latter code are shown in Table IX.

	Error Code (2)
00	No Error
01	Memory Error
10	Processor Error
11	Undetermined Error

BOSS uses the CPE's code to determine which subroutine to branch to in response to the processors status.

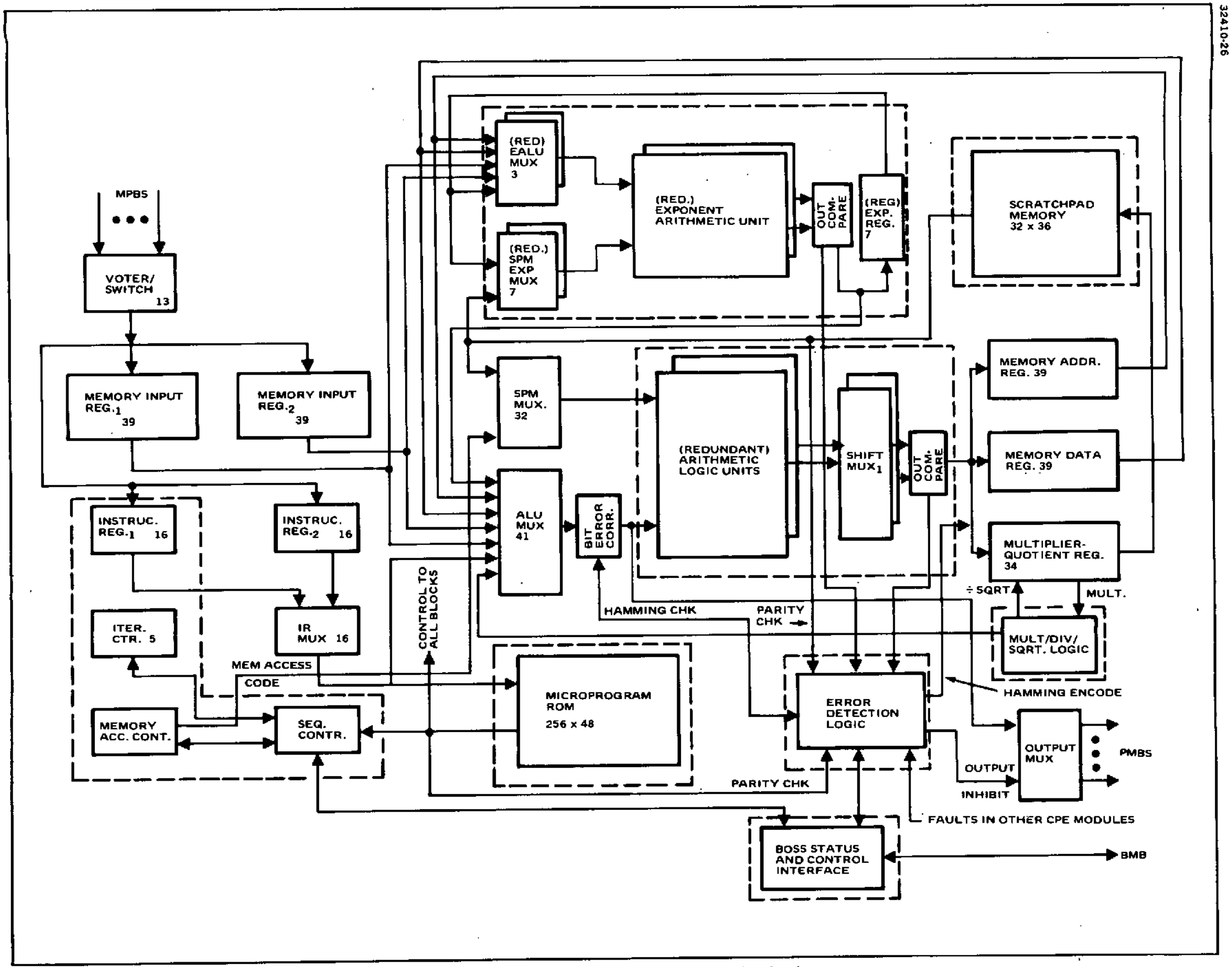
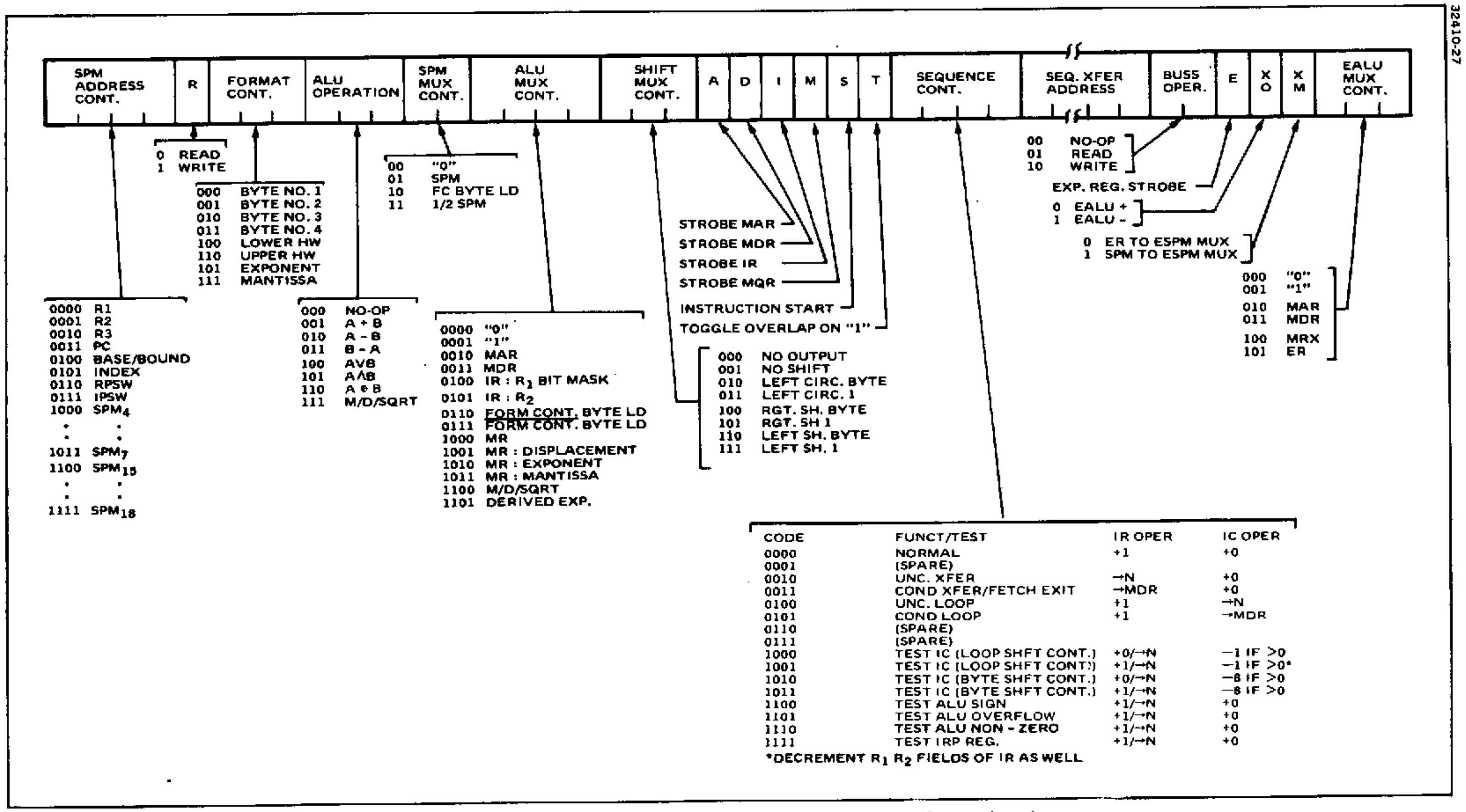


Figure 10. ARMMS CPE Functional Block Diagram



0000	R1
0001	R2
0010	R3
0011	PC
0100	BASE/BOUND
0101	INDEX
0110	RPSW
0111	IPSW
1000	SPM_4
⋮	⋮
1011	SPM_7
1100	SPM_{15}
⋮	⋮
1111	SPM_{18}

0	READ
1	WRITE

000	BYTE NO. 1
001	BYTE NO. 2
010	BYTE NO. 3
011	BYTE NO. 4
100	LOWER HW
110	UPPER HW
101	EXPONENT
111	MANTISSA

000	NO-OP
001	A + B
010	A - B
011	B - A
100	AVB
101	A∧B
110	A ⊕ B
111	M/D/SQRT

00	"0"
01	SPM
10	FC BYTE LD
11	1/2 SPM

0000	"0"
0001	"1"
0010	MAR
0011	MDR
0100	IR : R_1 BIT MASK
0101	IR : R_2
0110	FORM CONT. BYTE LD
0111	$\overline{\text{FORM CONT.}}$ BYTE LD
1000	MR
1001	MR : DISPLACEMENT
1010	MR : EXPONENT
1011	MR : MANTISSA
1100	M/D/SQRT
1101	DERIVED EXP.

000	NO OUTPUT
001	NO SHIFT
010	LEFT CIRC. BYTE
011	LEFT CIRC. 1
100	RGT. SH. BYTE
101	RGT. SH 1
110	LEFT SH. BYTE
111	LEFT SH. 1

CODE	FUNCT/TEST	IR OPER	IC OPER
0000	NORMAL	+1	+0
0001	(SPARE)		
0010	UNC. XFER	→N	+0
0011	COND XFER/FETCH EXIT	→MDR	+0
0100	UNC. LOOP	+1	→N
0101	COND LOOP	+1	→MDR
0110	(SPARE)		
0111	(SPARE)		
1000	TEST IC (LOOP SHFT CONT.)	+0/→N	−1 IF >0
1001	TEST IC (LOOP SHFT CONT.)	+1/→N	−1 IF >0*
1010	TEST IC (BYTE SHFT CONT.)	+0/→N	−8 IF >0
1011	TEST IC (BYTE SHFT CONT.)	+1/→N	−8 IF >0
1100	TEST ALU SIGN	+1/→N	+0
1101	TEST ALU OVERFLOW	+1/→N	+0
1110	TEST ALU NON - ZERO	+1/→N	+0
1111	TEST IRP REG.	+1/→N	+0

*DECREMENT R_1 R_2 FIELDS OF IR AS WELL

00	NO-OP
01	READ
10	WRITE

0	EALU +
1	EALU -

0	ER TO ESPM MUX
1	SPM TO ESPM MUX

000	"0"
001	"1"
010	MAR
011	MDR
100	MRX
101	ER

Figure 11. CPE Microprogram Memory Organization

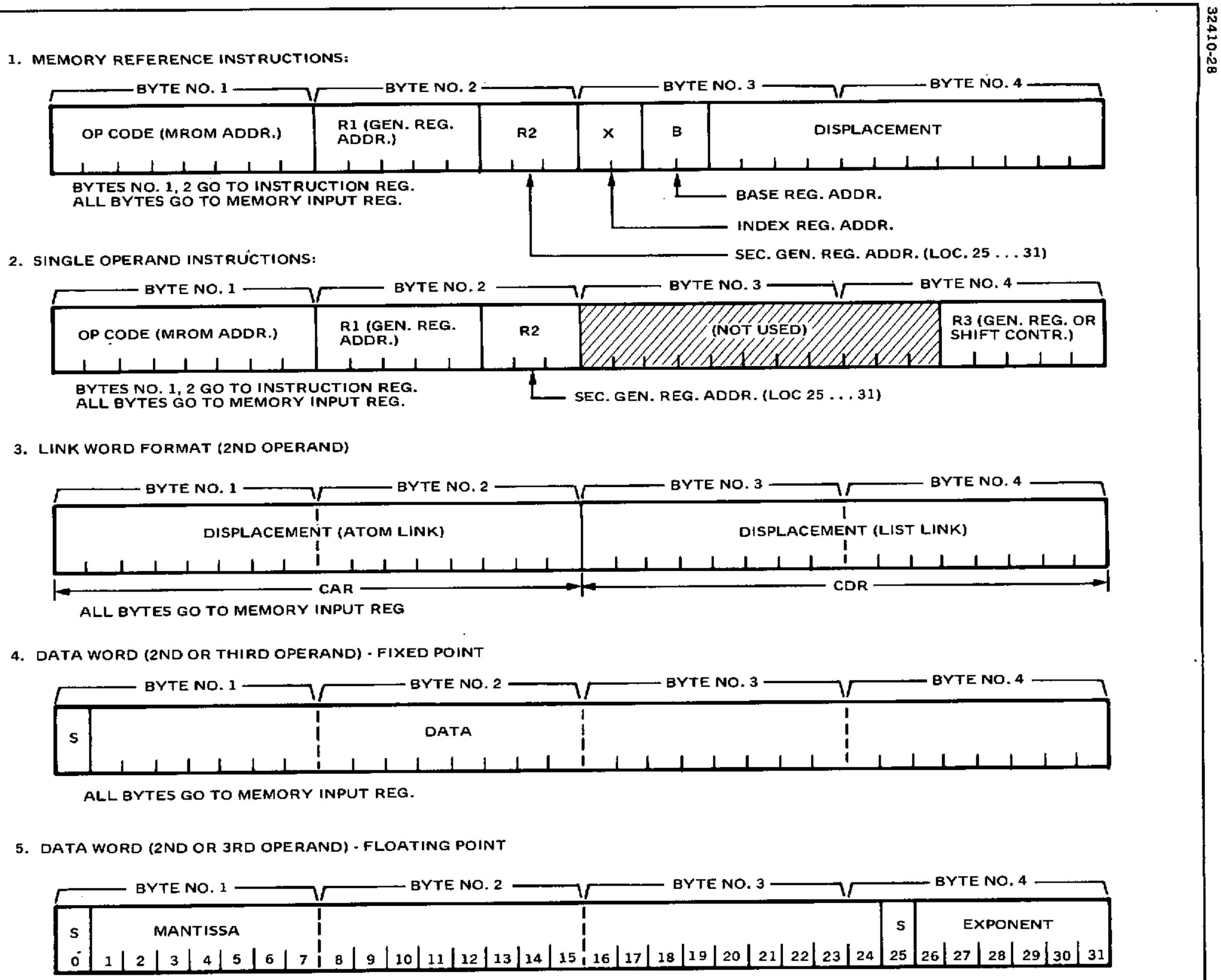

32410-28
1. MEMORY REFERENCE INSTRUCTIONS:
BYTE NO. 1
BYTE NO. 2
BYTE NO. 3
BYTE NO. 4
OP CODE (MROM ADDR.)
R1 (GEN. REG. ADDR.)
R2
X
B
DISPLACEMENT
BYTES NO. 1, 2 GO TO INSTRUCTION REG.
ALL BYTES GO TO MEMORY INPUT REG.
BASE REG. ADDR.
INDEX REG. ADDR.
SEC. GEN. REG. ADDR. (LOC. 25 . . . 31)
2. SINGLE OPERAND INSTRUCTIONS:
OP CODE (MROM ADDR.)
R1 (GEN. REG. ADDR.)
R2
(NOT USED)
R3 (GEN. REG. OR SHIFT CONTR.)
BYTES NO. 1, 2 GO TO INSTRUCTION REG.
ALL BYTES GO TO MEMORY INPUT REG.
SEC. GEN. REG. ADDR. (LOC 25 . . . 31)
3. LINK WORD FORMAT (2ND OPERAND)
DISPLACEMENT (ATOM LINK)
DISPLACEMENT (LIST LINK)
CAR
CDR
ALL BYTES GO TO MEMORY INPUT REG
4. DATA WORD (2ND OR THIRD OPERAND) - FIXED POINT
S
DATA
ALL BYTES GO TO MEMORY INPUT REG.
5. DATA WORD (2ND OR 3RD OPERAND) - FLOATING POINT
S
MANTISSA
S
EXPONENT
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Figure 12. CPE Instruction and Data Formats

TABLE VIII. CPE INSTRUCTION SET

The CPE instruction set is derived from a subset of the BOSS instruction set with the addition of more arithmetic instructions.

BOSS Instructions not Required:	
LIT	Load and Start Interval Timer
RSC	Read System Clock Register
COM	Command Module via BMB
INM	Interrogate Module via BMB
SIM	Set Interrupt Mask
Added CPE Instructions:	
SRD	Shift Right Double
SLD	Shift Left Double
MPY	Multiply
DVD	Divide
SQR	Square Root
DAD	Add Double
DSB	Subtract Double
FAD	Floating Point Add
FSB	Floating Point Subtract
FMP	Floating Point Multiply
FPV	Floating Point Divide

Total BOSS Instructions	58
Total BOSS not Required	-5
Added CPE Instructions	11
Total CPE Instructions	64

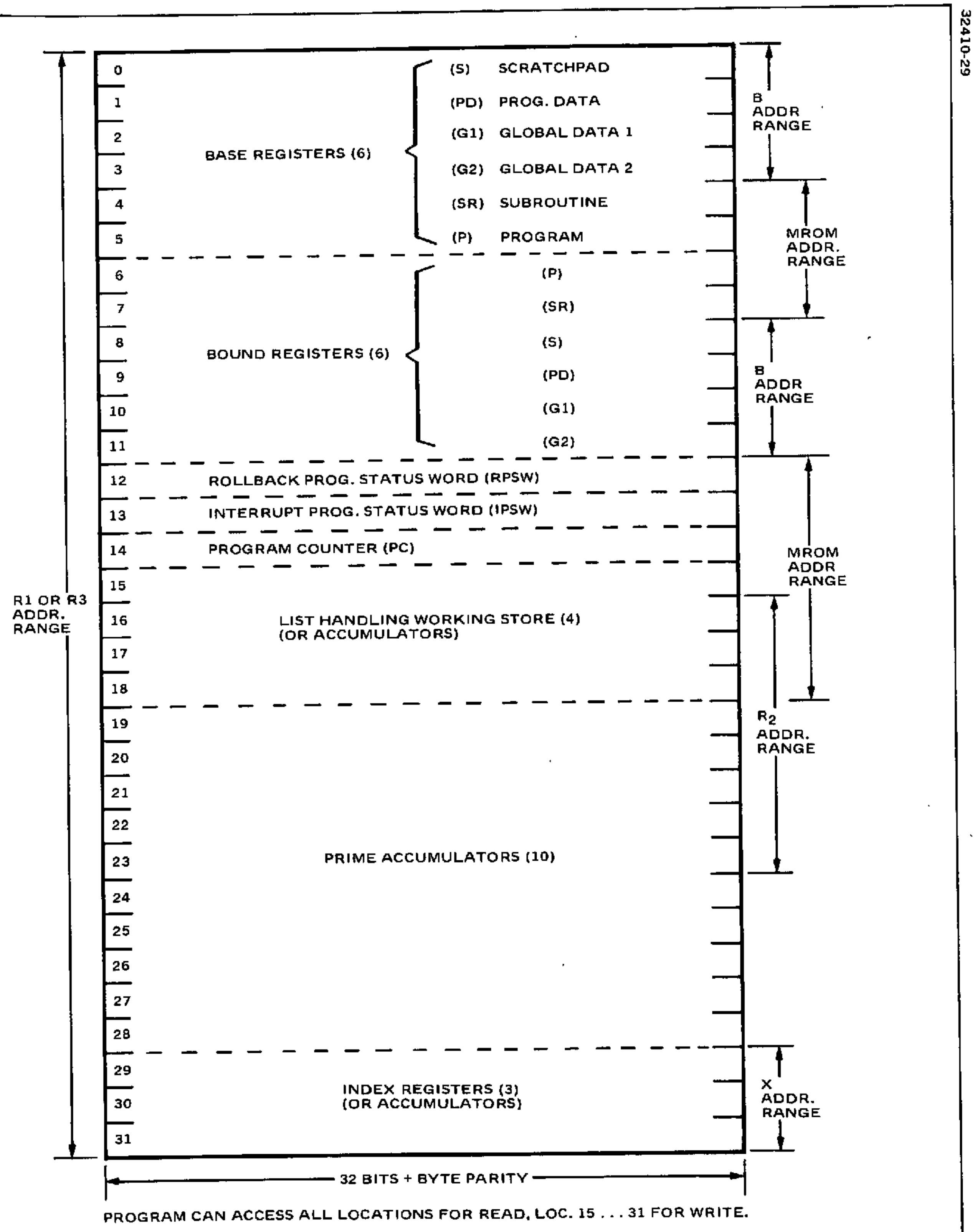


Figure 13. CPE Scratchpad Memory Organization

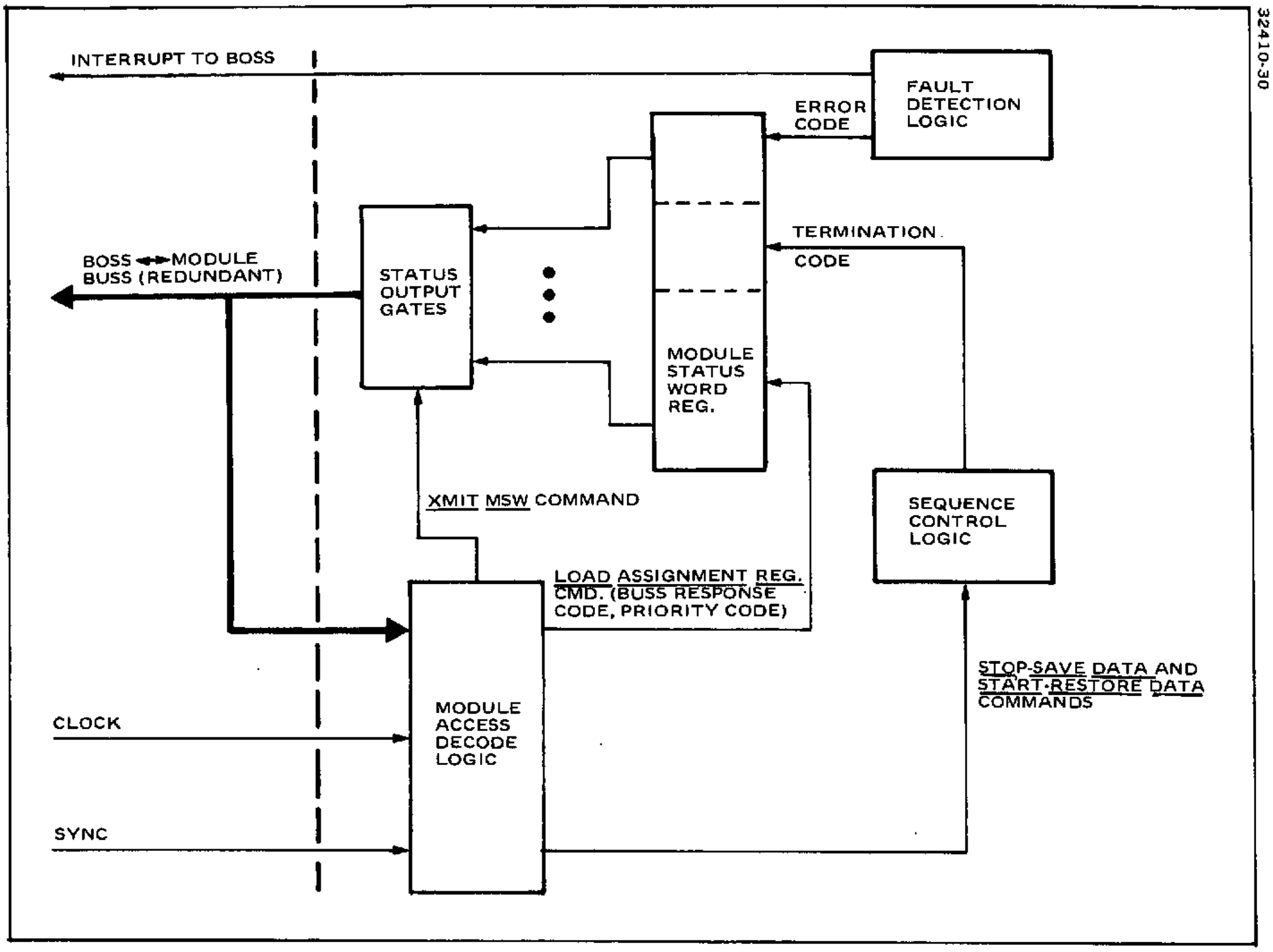

INTERRUPT TO BOSS
FAULT DETECTION LOGIC
ERROR CODE
TERMINATION CODE
BOSS ↔ MODULE BUSS (REDUNDANT)
STATUS OUTPUT GATES
MODULE STATUS WORD REG.
XMIT MSW COMMAND
SEQUENCE CONTROL LOGIC
LOAD ASSIGNMENT REG. CMD. (BUSS RESPONSE CODE, PRIORITY CODE)
STOP-SAVE DATA AND START-RESTORE DATA COMMANDS
CLOCK
SYNC
MODULE ACCESS DECODE LOGIC
32410-30

Figure 14. CPE Module Interface for BOSS Control and Status Communication

4.5 ARMMS IOP Register Level Design and Reliability Study

A register level design and reliability analysis have been completed for the ARMMS IOP along with a study of IOP/CPE/BOSS commonality. The ARMMS Executive software design described in this report requires an IOP with capabilities far greater than the standard channel - control unit configuration. The resulting IOP presents computing capabilities approximating those of BOSS, less the reconfiguration and list processing features. Coupled to this minimal computing element is the high-speed selector type channel that will support the CVT Data Bus featuring a sixteen bit wide data path.

The channel and processing unit are combined to form the IOP sharing a common interface to memory. The two units form a pair with the channel being a slave only to its own processing unit through a system of interrupts allowing concurrent processing and data transfer operations.

TABLE IX. PROCESSOR REQUEST COMMANDS

1.	JOB SCHEDULE
2.	JOB TERMINATE
3.	JOB ABEND (may or may not be combined with job terminate)
4.	JOB CANCEL
5.	TASK SCHEDULE
6.	TASK TERMINATE
7.	TASK ABEND (may or may not be combined with task terminate)
8.	TASK CANCEL
9.	TASK STATUS
10.	SYSTEM SUBROUTINE CALL
11.	SYSTEM SUBROUTINE COMPLETION
12.	LOCK VARIABLE
13.	UNLOCK VARIABLE
14.	GETMAIN
15.	FREEMAIN
16.	TIME OF DAY
17.	WAIT CALL
18.	EVENT SET
19.	ALERT CALL
20...32	SPARE

The channel presents features over and above the standard selector channel. Memory is protected during all transfers with a set of channel base and bounds registers as well as those controlling the processing unit. An additional feature is the channel index allowing cyclic operation within the channel program without intervention by the processing unit.

4.5.1 IOP Commonality With CPE

The processing unit of the IOP is a subset of the CPE design providing maximum commonality with the CPE. The channel, however, has less commonality due to its nature as an interface rather than a processing unit. Still, commonality is kept in 33% of the channel through use of comparable ALU and scratchpad memory LSICs. Since the channel logic represents only about 33% of the IOP total, the effect of special channel logic has less overall significance. The information of Table IX shows the IOP having 45 LSI devices, 36 of which are found in the CPE giving a 80% commonality with existing CPE logic.

4.5.2 IOP Reliability Analysis

By operating the IOP in at least a duplex mode, failures in most IOP logic will be detected including those in the ALU and control logic blocks. Hamming and Parity logic is provided in order to check the main memory using techniques common with the CPE and BOSS modules. Memories internal to the IOP are protected by a parity system allowing testing for odd numbers of stuck bits. Table X lists IOP failure modes and suggested corrective action. Failure analysis leads to the following results:

1. If an IOP is replaced at the end of a task in which it fails, and software is capable of switching the IOP output to a redundant memory port if the primary memory port fails, the IOP has the following reliability characteristics:

Logic Failure rate/10^6 hours	$\simeq$1.065
Simplex mode coverage	$\simeq$59%
Duplex or TMR coverage	$\simeq$100%
Failures Maskable in Simplex	$\simeq$12%
Failures Maskable in Duplex	$\simeq$59%
Failures Maskable in TMR	$\simeq$100%

2. As with the CPE power supplies and bus interface electronics failure rates are less than 10% of the logic failure rate.

3. Approximately 10% of IOP logic is devoted to failure detection and correction in the baseline IOP design. This logic detects most memory module failures and more than half of those within the IOP.

4. Assuming 4 IOP's are initially flown, the probability of different numbers of IOP's remaining operational within a 5 year mission is shown below.

TABLE X. IOP FAILURE MODES

Components Failing	Result	Failure/ 10^6 Hr	Corrective Action
Input mux or voter/ switch	Triple-bit error	0.064	Detect with H-P code-inhibit output
Mem in, addr, data reg, instr reg, output and ALU muxes	Single bit error	0.125	Detect and mask with H-P code
Scratchpad memory (SPM), MQR	Single bit error	0.195	Detect with parity check-inhibit output, est coverage 0.95
Arithmetic logic unit	Multiple bit error	0.178	Detect in duplex, mask in TMR simplex coverage = 0
Microprogram ROM (MROM)	Control bit error	0.125	Detect with parity check-inhibit output, est coverage 0.9
Instruc reg and mux	SPM or MROM addr bit error	0.031	Detect by comparison with mem input reg — inhibit output
Iter ctr, seq and mem access contr BOSS status and contr interface	Improper execution loss of sync	0.040	Detect in duplex, mask in TMR simplex coverage = 0
Error detection logic	False error indication	0.108	Inhibit output
Channel registers	Single bit error	0.099	Detect in duplex, mask in TMR simplex coverage = 0
Channel control	Improper execution loss of sync	0.100	Detect in duplex, mask in TMR simplex coverage = 0
		1.065	

NOTE: Coverage = 1.0 unless otherwise noted.

Number of IOP's Operational	Probability After 5 Years
4	0.8482
≥3	0.9908
≥2	0.9997

This shows that only one spare above the TMR Configuration need be flown but the possibility of one failing is significant and the system should reconfigure gracefully to accept this.

5. As with the CPE, errors are masked if possible or else program rollback is attempted. Success of either will cause inhibition of immediate interrupt to BOSS until task completion.

4.5.3 IOP Register Level Design

The IOP register level design is shown in Figure 15. It consists of two basically independent units separated by the dashed line. Above is the processing unit capable of executing a stored program with a repertoire and structure similar to partition 'A' of the BOSS module. A slave to this unit is the channel shown at the bottom of the Figure. It is capable of executing its own channel program consisting of a string of I/O commands chosen from the channel repertoire. The execution is begun at the command of the processing unit and may continue concurrently with further processing unit operation. Both the channel and the processing unit share a common memory interface on a cycle-stealing basis with the channel having highest priority.

The processing unit shares similarities between both the CPE and BOSS modules. In the system it performs as would another CPE in relation to BOSS. However, the instruction set shown in Table XI is close to that of BOSS with added I/O instructions. This reduces the processing unit complexity by eliminating floating point and complex arithmetic instructions of the CPE. These instructions are not used in I/O operations and their elimination greatly reduces the IOP complexity while still providing efficient processing support.

The IOP instructions formats are identical to those of BOSS and CPE with the addition of the special I/O instruction format. This is shown as F3 in Figure 16 along with F1 and F2, the memory and register reference instructions respectively. These formats are identical to these used by the BOSS and CPE modules.

The processor of the IOP is a reduced version of the CPE. Eliminated are the exponent ALU and associated MUX's, the multiplier-quotient register and its multiply, divide, and square root logic, instruction - data look-ahead registers, and redundant ALU. This results in a slightly modified MROM format as shown in Figure 17. The format control, ALU operation, and ALU MUX control fields retain their width but the extent of code usage is reduced in the IOP. The strobe MQR and toggle overlap bits are not required due to the absence of these features as are the last six bits of exponent strobe and MUX control. Added to the CPE format are the I/O request and interrupt controls primarily used for requesting and responding to channel operations respectively.

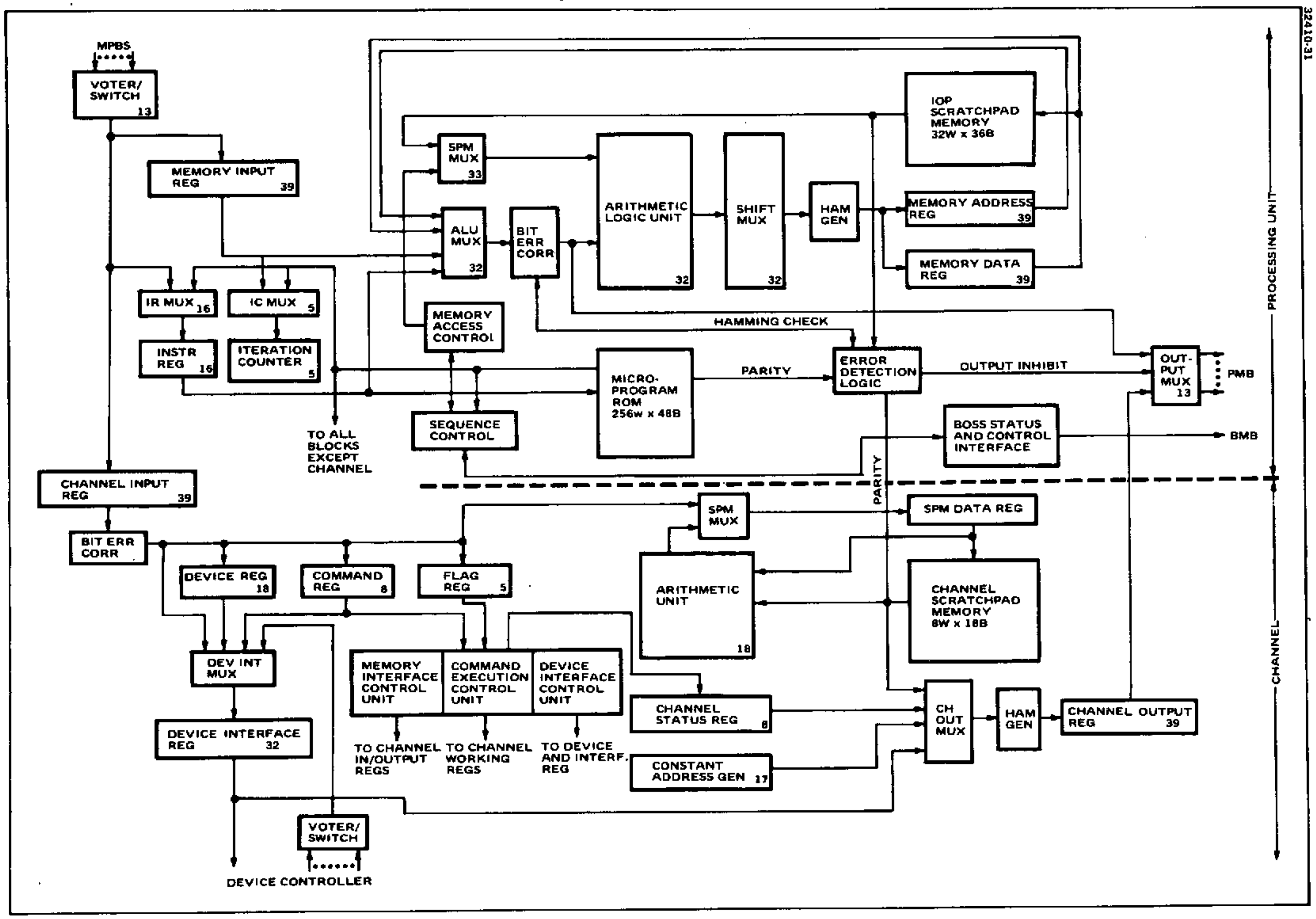


Figure 15. ARMMS IOP Functional Block Diagram

TABLE XI. IOP INSTRUCTION SET

The IOP instruction set is derived from a subset of the BOSS instruction set with the addition of I/O oriented instructions.

BOSS Instructions not Required:	
LIT	Load and Start Interval Timer
RSC	Read System Clock Register
COM	Command Module via BMB
INM	Interrogate Module via BMB
	List Manipulation Instructions
Added IOP Instructions:	
SIO	Start Input or Output
CLR	Clear Channel
HLT	Halt Channel
ICA	Interrogate Command Address
IST	Interrogate Status

Total BOSS Instructions	58
Total BOSS Not Required	-8
Added IOP Instructions	5
Total IOP Instructions	55

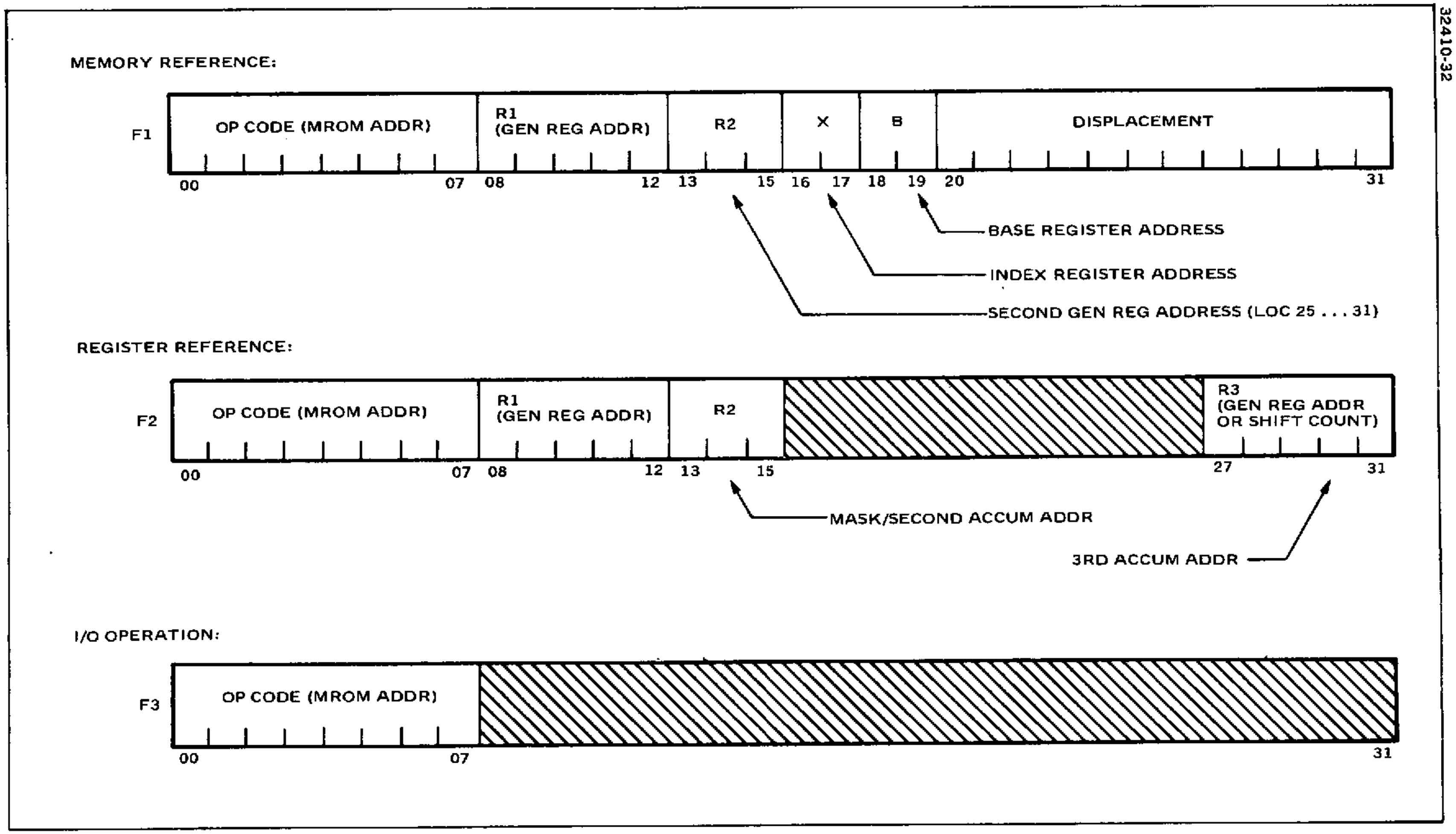
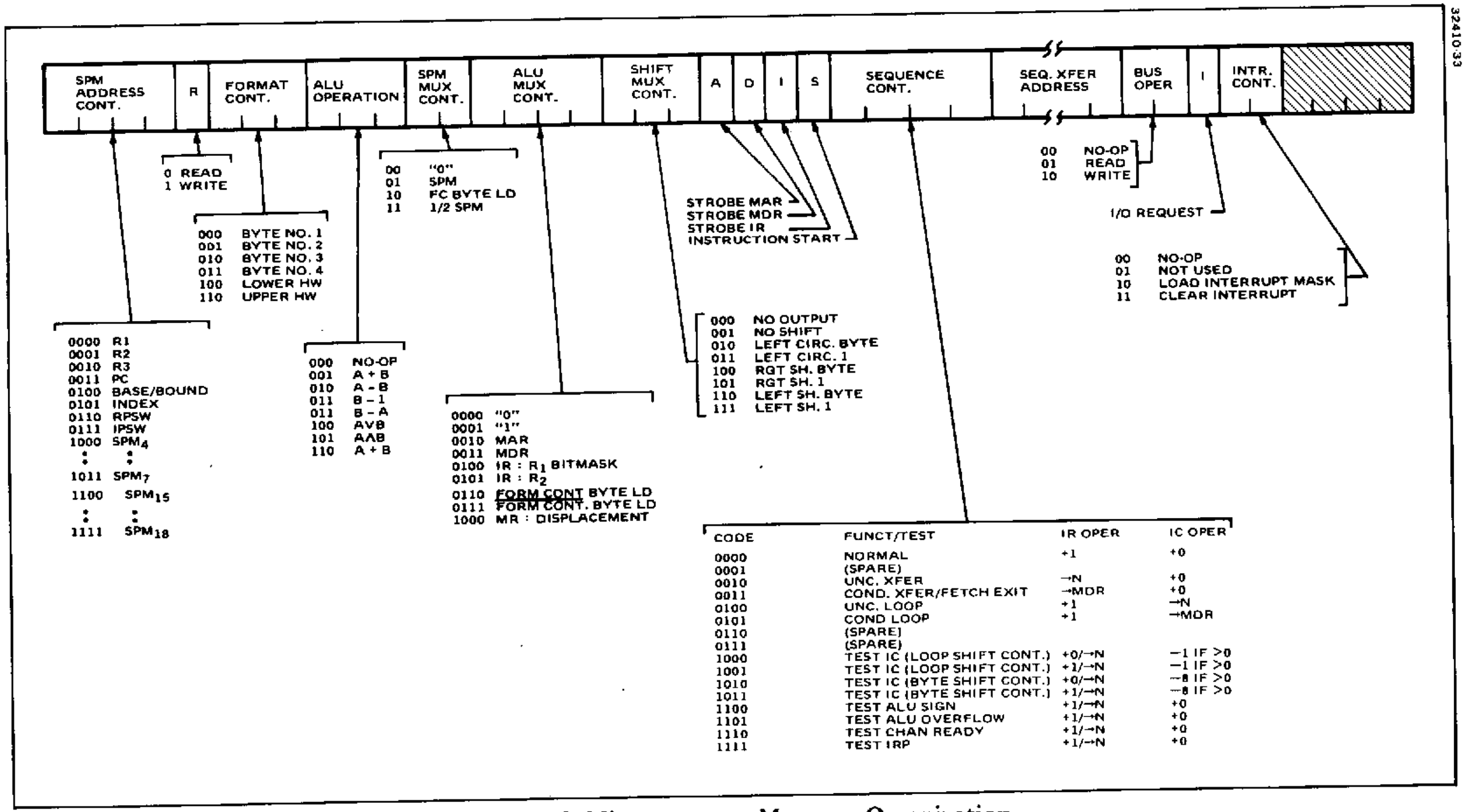


Figure 16. IOP Instruction Formats



CODE	FUNCT/TEST	IR OPER	IC OPER
0000	NORMAL	+1	+0
0001	(SPARE)		
0010	UNC. XFER	→N	+0
0011	COND. XFER/FETCH EXIT	→MDR	+0
0100	UNC. LOOP	+1	→N
0101	COND LOOP	+1	→MDR
0110	(SPARE)		
0111	(SPARE)		
1000	TEST IC (LOOP SHIFT CONT.)	+0/→N	−1 IF >0
1001	TEST IC (LOOP SHIFT CONT.)	+1/→N	−1 IF >0
1010	TEST IC (BYTE SHIFT CONT.)	+0/→N	−8 IF >0
1011	TEST IC (BYTE SHIFT CONT.)	+1/→N	−8 IF >0
1100	TEST ALU SIGN	+1/→N	+0
1101	TEST ALU OVERFLOW	+1/→N	+0
1110	TEST CHAN READY	+1/→N	+0
1111	TEST IRP	+1/→N	+0

Figure 17. IOP Microprogram Memory Organization

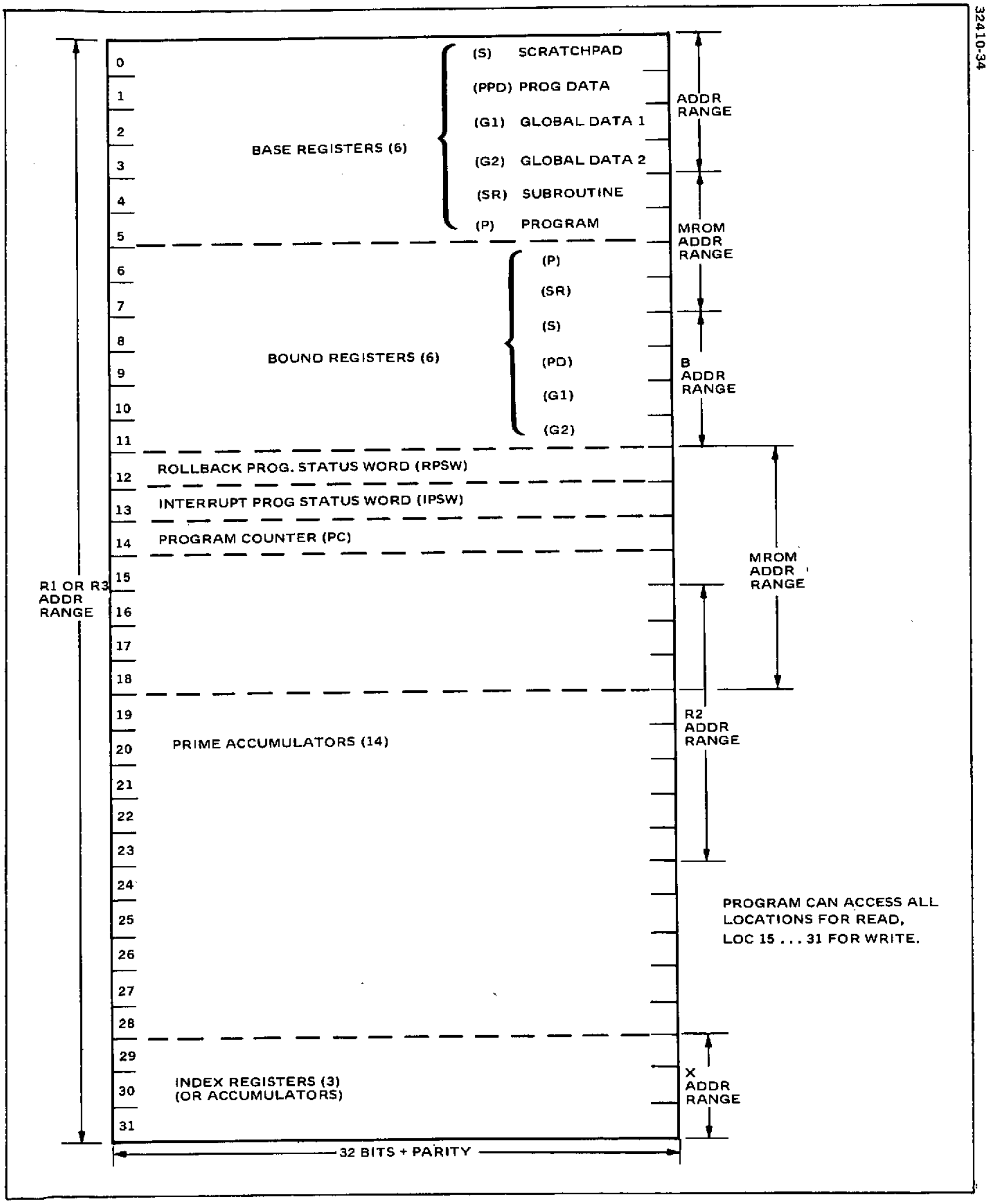


Figure 18. IOP Scratchpad Memory Organization

The IOP scratchpad memory is mapped according to Figure 18. This is identical to the CPE scratchpad memory except for the absence of list handling references since these instructions do not fall into the IOP repertoire.

Once initiated by the processor via I/O instructions, the channel executes from a set of eight channel commands shown in Table XII. More than one of these commands may be chained together forming a channel program. The commands conform to one of three formats shown in Figure 19. The double word commands each have an eight bit command code in the first word, and the remainder of the word supplies the memory reference address if required. The second word of the command presents a set of flags for controlling the mode of execution, and count information for data transfers and Load Index. The second word may also contain a status mask in the case of the Transfer On Status command.

The need for special communication paths from processor to channel is minimized through use of fixed core memory locations accessible by both units. These locations are summarized in Figure 20 which shows the Command Address Word (CAW) and the Channel Status Word (CSW). The former is generated by the processor during I/O initiation while the latter is channel generated status information. Particular bits of the status information are in standard form and described below.

Figure 21 shows the organization of the eight words of channel scratchpad memory used for efficiently storing bookkeeping data for the channel. Command and data addresses together with their bases and bounds form the first six words while the data count and channel index reside in the remaining two locations.

TABLE XII. IOP CHANNEL COMMANDS

The channel is capable of executing a chained channel program independent of the IOP main program.

Channel Command Set;		Format
INP	Input Data	F4
OUT	Output Data	F4
CNT	Control	F4
SNS	Sense	F4
TRS	Transfer on Status	F5
TRA	Transfer Unconditional	F5
TRX	Transfer on Index	F5
LDX	Load Index	F6

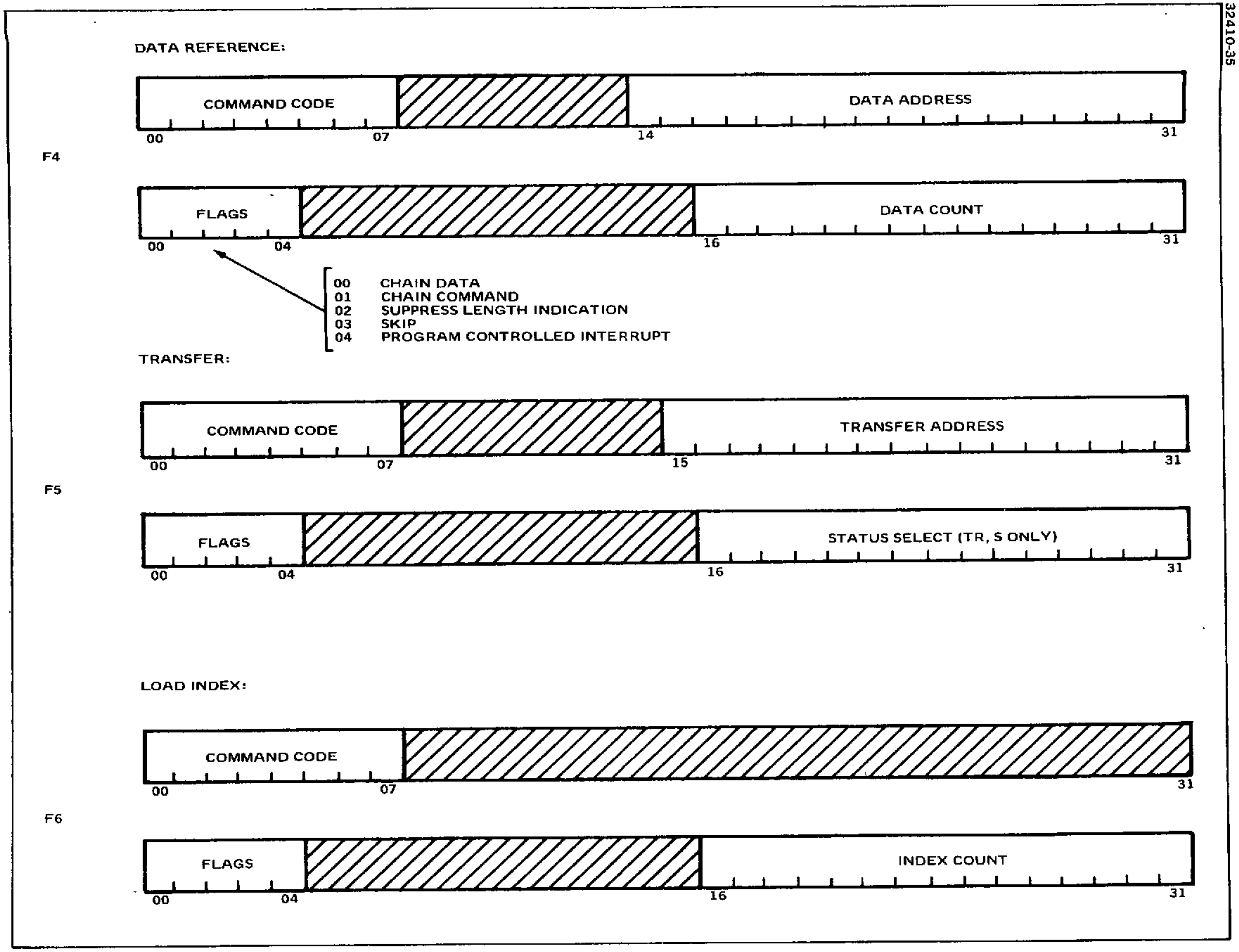
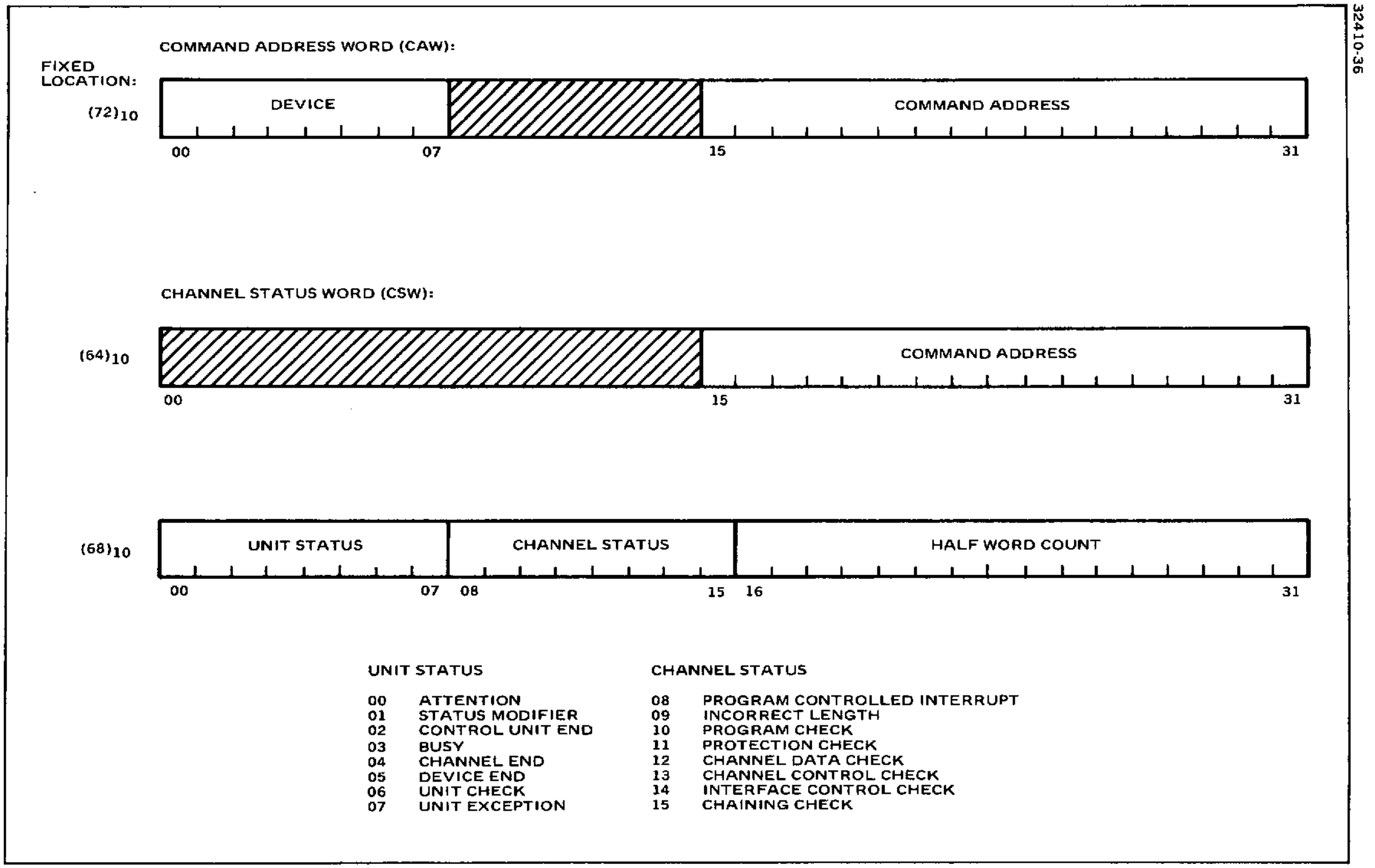


Figure 19. I/O Channel Command Word (CCW) Formats



UNIT STATUS		CHANNEL STATUS	
00	ATTENTION	08	PROGRAM CONTROLLED INTERRUPT
01	STATUS MODIFIER	09	INCORRECT LENGTH
02	CONTROL UNIT END	10	PROGRAM CHECK
03	BUSY	11	PROTECTION CHECK
04	CHANNEL END	12	CHANNEL DATA CHECK
05	DEVICE END	13	CHANNEL CONTROL CHECK
06	UNIT CHECK	14	INTERFACE CONTROL CHECK
07	UNIT EXCEPTION	15	CHAINING CHECK

Figure 20. IOP Fixed Memory Control Word Formats

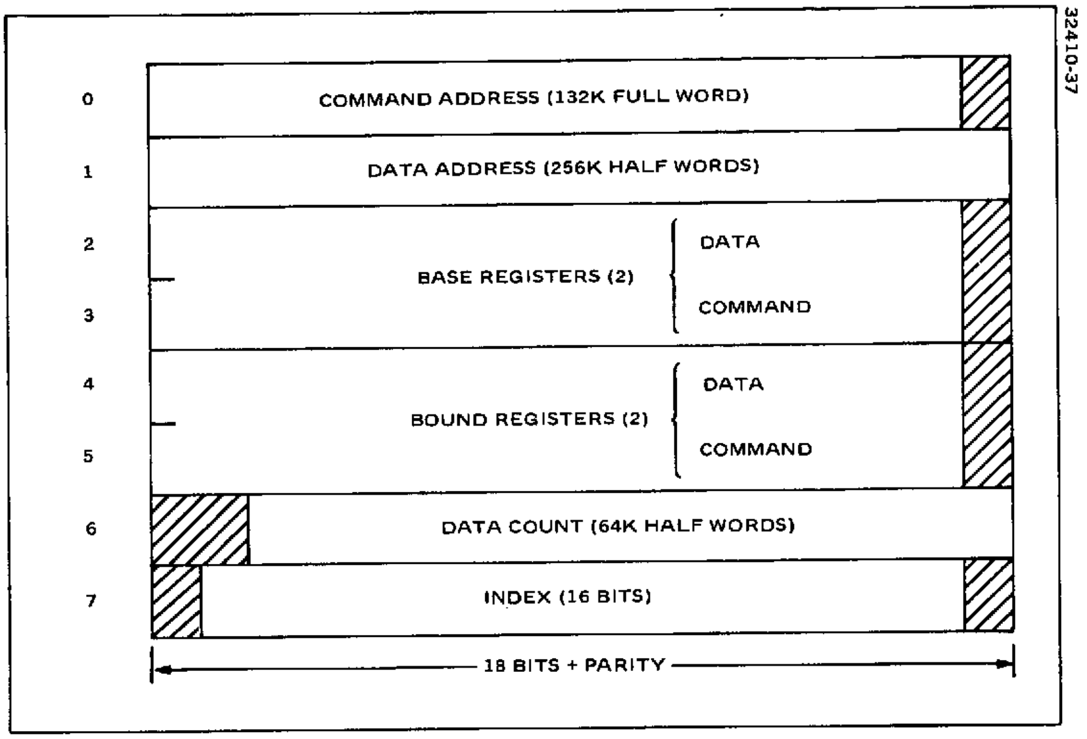


Figure 21. Channel Scratchpad Memory Organization

4.6 SUMC LSI Module Study

A study to assess the applicability of the existing SUMC LSI Module set to ARMMS and of ARMMS reliability enhancement techniques to SUMC has been performed. Three approaches to adding controlled redundancy to increase a SUMC computer's life time are available:

1) Use redundant SUMC processors and main memory units with voters and/or comparators provided at unit outputs.

2) Apply redundancy and error coding at the LSI module level by adding additional LSI modules but minimizing changes to existing modules. or,

3) Apply redundancy and error coding within the LSI modules.

In the CPE Register Level Design topic of this report alternative 3) was followed with no restrictions being assumed on the logic due to other SUMC related efforts. This approach has led to an efficient reliable logic design for an ARMMS processor. However its LSI modules are not compatible with existing SUMC LSI modules and it is useful to assess the cost to ARMMS in terms of reliability and performance of establishing commonality with the SUMC modules.

Alternative 1) above is the traditional approach to reliability enhancement. It has been applied to whole computers by comparing I/O signals or to processors and memories by comparing outputs of redundant units within a single computer. This method requires at least an 100% increase in complexity for detection and a 200% increase in complexity for real-time correction of erroneous computations. The exact increase would depend on the complexity of the voter/ comparitor units over and above the duplication or triplication of the processors and memories. It is comparatively simple in terms of design and would require minimum change to the existing SUMC processor but it is costly in terms of total hardware complexity.

Alternative 2) becomes attractive if it is possible to detect or correct most unit errors using less redundancy within a unit than would have been required to duplicate that unit. As might be expected some portions of a processor are more amenable to error isolation than others. A major objective of this study is to suggest specific techniques and associated complexities for error isolation in each section of SUMC's architecture. It is expected that the trade-offs as to how much error detection and correction logic would be placed inside a processor would be mission dependent and that in an ARMMS computer where simplex, duplex and TMR processing modes are available it would not be necessary to detect all possible errors within a single processor since programs requiring this degree of detection could be run in a duplex or TMR mode.

In addition to the trade-offs between adding controlled redundancy and redesigning modules vs adding modules or units it is necessary to perform trade-offs between alternate ways of performing required functions since some mechanizations require either less hardware or hardware in which errors are more readily isolated. A reliable design should first attempt to minimize each unit's failure rate by minimizing its complexity for a given level of performance and second attempt to maximize the percentage of errors that can be detected if they occur, assuming that the computer is considered to have failed if it either cannot perform a required computation correctly or unknowingly performs an erroneous computation in a critical program.

SUMC consists of 5 major building blocks: 1) The Scratchpad Memory (SPM), containing 64 words of 32 data bits each, includes general and floating point registers, program status information, and working and mask registers used for program instruction execution; 2) the Arithmetic Logic Unit (ALU), presently consisting of three multiplexers and two parallel arithmetic units including fast carry logic, selects data sources and performs required logical or arithmetic operations; 3) the Multiplexer-Register Unit (MRU), consisting of three multiplexers and three registers, is used to transfer data from the ALU to the SPM or to the main memory modules and to retain the results of intermediate microinstructions during microprogram execution; 4) the Floating Point Unit (FPU), consisting of a 32 bit multiplexer, an 8 bit Exponent Arithmetic Logic Unit (EALU), and an 8 bit Exponent Register (ER), is used for the solution and normalization of floating-point operations; and 5) the Control Unit (CU) decodes program instructions and provides the ALU, SPM, MRU, and FPU control signals required for their execution.

The major units within the CU and their functions are: 1) the Instruction Register (IR) which holds the instruction being executed; 2) the Instruction Address Read-Only-Memory (IAROM) containing 256 words of 22 bits each, which is addressed by the executed instruction's operation code and whose output provides the starting address for the microprogram which must be executed to perform that instruction and format control information associated with the instruction; 3) the Sequence Control Unit (SCU) which addresses the Microprogram ROM (MROM) and contains a loadable iteration counter and a MROM address register/counter whose contents are modified during microinstruction execution to provide microprogram sequencing; and 4) a MROM, having 1024 words of 72 bits each, contains the prestored sequences of microinstructions required to fetch and execute program instructions, initiate IOP and main memory accesses, and respond to external interrupts.

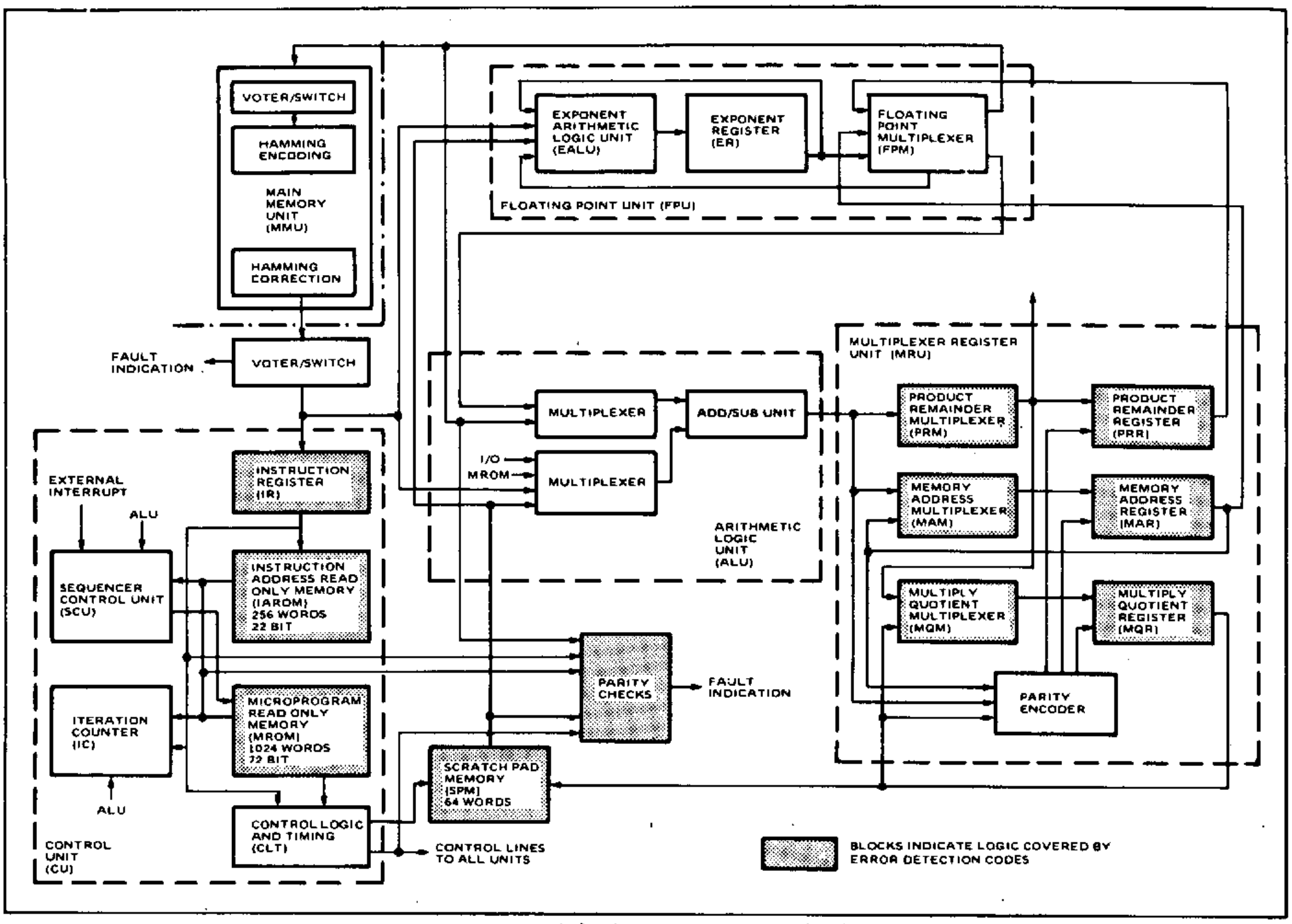
SUMC functional units overlap LSI module boundaries somewhat. The ALU and SPM represent groups of ALU and SPM modules. The MRU modules accomplish all MRU functions plus IR and ER functions. The FPU is made up of Floating Point multiplexer modules plus ALU and portions of MRU modules. The CU is made up of Sequence Control Unit, Function Control Unit, and Data Control Unit modules plus groups of IAROM and MROM modiles.

Four methods of enhancing SUMC reliability, both for ARMMS and in other applications have been investigated: 1) There are several areas where it should be possible to reduce SUMC complexity without significantly reducing performance; 2) Failures in about 70% of SUMC logic can be detected through the use of coding techniques at an increase in complexity of about 10% in this portion of the SUMC logic; 3) Failures in the remaining SUMC logic can also be detected but this requires increasing these portions of the logic by over 100 percent; 4) Hamming codes and voter/switch techniques in conjunction with spare modules can be used to detect and/or correct failures in the SUMC computer's main memory unit and in the portions of SUMC logic not covered in 2) above. Figure 22 shows a block diagram of the SUMC processor with error detection logic added. The cross-hatched blocks are the ones in which errors can be easily detected. The floating point multiplexer also falls into this category during fixed point instructions (i.e., when it is simply used for transferring fixed point data).

4.6.1 Speed Enhancement Through Modification of SUMC Logic

An evaluation of the speed limitations of SUMC in ARMMS determined that the biggest speed bottleneck is likely to be the SUMC logic itself. Assuming either low-power MSI Schottky TTL (1973 time frame) or projected LSI CMOS using a silicon on sapphire technology (in the late 70's) maximum microinstruction clock rates would be on the order of 4 MHz. Data bus transmission from main memory to processor would be accomplished at twice this rate and main memory cycle times on the order of 800 nsec should be easily attainable at low power using plated wire techniques – hence these two areas should not be a problem. Using these numbers, the average instruction requires 3.5 μsec to execute (examples: Add $\approx$ 3 μsec, Divide $\approx$ 9.5 μsec, jump $\approx$ 2 μsec).

An average speed increase of from 30 to 40% can be achieved by instruction overlap – i.e., fetching the next instruction while executing the present instruction thus saving memory access and bus transfer time. In the best case two overlapped cycles correspond to one non-overlapped cycle and a program can be executed twice as fast as before. This occurs when a program consisting of short instructions such as LOAD and ADD is accessing a memory with no contention from other programs. The worst cases occur on JUMP instructions, STOREs of data generated in the immediately preceeding instruction, or when



VOTER/SWITCH
HAMMING ENCODING
MAIN MEMORY UNIT (MMU)
HAMMING CORRECTION
FAULT INDICATION
VOTER/SWITCH
EXPONENT ARITHMETIC LOGIC UNIT (EALU)
EXPONENT REGISTER (ER)
FLOATING POINT MULTIPLEXER (FPM)
FLOATING POINT UNIT (FPU)
MULTIPLEXER REGISTER UNIT (MRU)
MULTIPLEXER
ADD/SUB UNIT
I/O
MROM
MULTIPLEXER
ARITHMETIC LOGIC UNIT (ALU)
PRODUCT REMAINDER MULTIPLEXER (PRM)
PRODUCT REMAINDER REGISTER (PRR)
MEMORY ADDRESS MULTIPLEXER (MAM)
MEMORY ADDRESS REGISTER (MAR)
MULTIPLY QUOTIENT MULTIPLEXER (MQM)
MULTIPLY QUOTIENT REGISTER (MQR)
PARITY ENCODER
EXTERNAL INTERRUPT
ALU
SEQUENCER CONTROL UNIT (SCU)
INSTRUCTION REGISTER (IR)
INSTRUCTION ADDRESS READ ONLY MEMORY (IAROM) 256 WORDS 22 BIT
ITERATION COUNTER (IC)
MICROPROGRAM READ ONLY MEMORY (MROM) 1024 WORDS 72 BIT
ALU
CONTROL UNIT (CU)
CONTROL LOGIC AND TIMING (CLT)
PARITY CHECKS
FAULT INDICATION
SCRATCH PAD MEMORY (SPM) 64 WORDS
CONTROL LINES TO ALL UNITS
BLOCKS INDICATE LOGIC COVERED BY ERROR DETECTION CODES
32410-38

Figure 22. Modified SUMC CPE Block Diagram

two programs both consisting of short instructions are in heavy contention for the same memory page. In these cases overlap becomes ineffective and the program runs at the same speed as it would have without overlap. The average speed increases noted have been verified by computer simulations performed by Don Taylor of Computer Sciences Corp. These speed increases allow reducing the average instruction execution time to 2.5 μsec at a 4 MHz microinstruction clock rate.

Instruction overlap logic should amount to about a 5% increase in complexity for ARMMS including increases in both the SUMC CPEs and the main memory modules. The added logic requirements include:

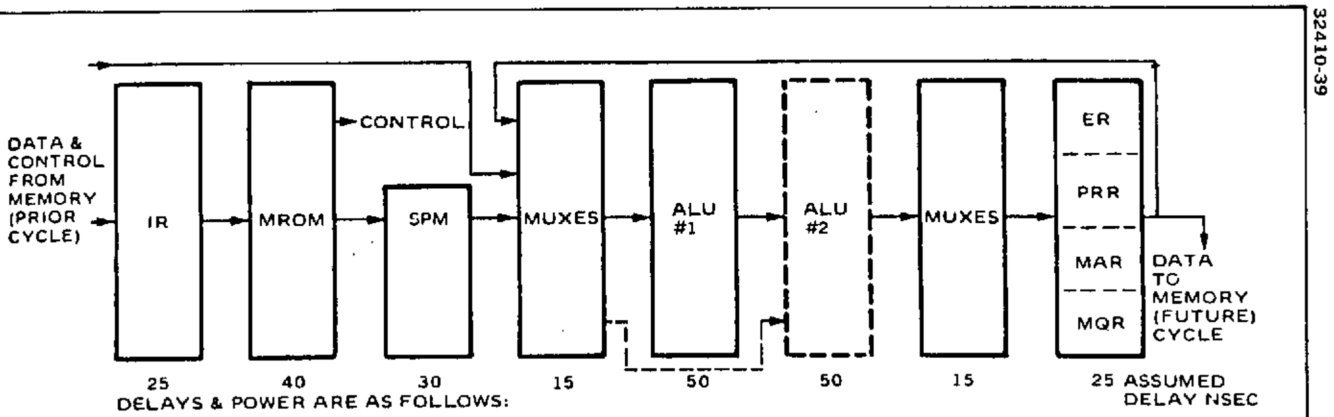
1. Logic to inhibit overlaps on JUMP and some STORE instructions.
2. Duplicated instruction registers to allow push-pull MROM access.
3. Memory address and data buffering.

Instruction overlap timing was discussed in detail on page 2-27 of the ARMMS Phase II report.

4.6.2 Reliability Enhancement through Modification of SUMC Logic

Once an instruction has been fetched it must follow the critical path shown in Figure 23, during the execution of each microinstruction step. Note that two adders are included in SUMC to speed up multiply, divide and square root operations. If only one adder were included in SUMC rather than the present two, the hardware would be reduced by about 10% (by 6 LSI modules or 1320 equivalent gates) and the clock rate could be increased by about 25% due to the decreased propagation delays, speeding up all operations except Multiply (M), Divide (D) and Square-Root (SQR) by 25%. The M, D, and SQR instructions would require approximately 70% more micro instructions than they do presently, hence they would take 26% longer to execute than presently. However, except for programs requiring large numbers of M, D, SQR operations, SUMC's speed would show a net increase (5% if all instructions are assumed equally likely to be executed). Only for programs with more than 25% multiply, divide, square-root instructions would any speed reduction be noted. These operations typically make up no more than 1 to 7% of an instruction mix and even tasks such as matrix inversions require only a 15% mix of these instructions. Removing one adder also reduces the amount of redundancy needed in the system since adders cannot be checked using the same error detecting/correcting coding techniques proposed for the rest of ARMMS and hence require duplication and comparing of outputs if their failures are to be detected. For these reasons the use of only one adder in SUMC is recommended.

A similar argument can be made for the floating-point multiplexer structure (with the exception of the operand and exponent encoders) which is necessary only for floating-point instructions and whose functions could be performed serially by SUMC's multiplexer-register module, slowing these instructions by



DELAYS & POWER ARE AS FOLLOWS:

MROM	[SN54187]		32 W
SPM	[SN5489]		12 W
ALU	[SN54LS181]	LOW POWER SCHOTTKY WITH HI-POWER SCHOTTKY LOOK-AHEAD LOGIC	2 W
OTHERS	[SN54LSXXX]	LOW POWER SCHOTTKY	8 W
TOTAL	250 NSEC		54 W

CMOS LSI SHOULD HAVE SIMILAR RATINGS BY LATE 70s.

Figure 23. Critical Path Through Baseline SUMC

an average of 20%. The reduction in the number of gate delays could again allow increasing the flock frequency yielding a net increase in speed as well as a 5% reduction in SUMC complexity (3 LSI modules or 712 equivalent gates). It is important to note that these two changes reduce the complexity of the portion of SUMC logic in which errors are costly to detect by over 50%.

Roughly half of SUMC's complexity lies in its internal semiconductor ROM's and SPM's. Hence serious consideration should be given to reducing the size of these memories in missions where this is possible. Use of firmware interrupt routines not requring 4 separate sets of SPM registers could reduce the SPM size. It was possible to reduce the word length in the ARMMS CPE's MROM by 33% without sacrificing performance. It should also be possible to implement a reasonable instruction set in fewer than 1024 MROM words. If 256 words or less are adequate for the MROM and system 360 machine language code compatibility is not required, the IAROM could also be eliminated with the MROM addressed directly from the instruction register. In the ARMMS CPE these changes resulted in a 75% reduction in semiconductor memory chip count (21 LSI modules) assuming a ROM size of 4096 bits and a SPM size of 256 bits. Even a less drastic reduction should improve SUMC reliability.

4.6.3 Enhancing SUMC Reliability Through the Addition of Error Detecting Codes

Since parity tests that are valid after shift operations can be constructed relatively simply it is possible to detect all odd numbers of errors in SUMC's semiconductor memory modules, and multiplexer-register unit modules, and about 40% of the failure modes of the floating-point multiplexer modules (if the latter modules are retained). The logic to accomplish these checks requires adding approximately 1100 gates in four additional LSI modules to SUMC. These added circuits detect errors in 40 current SUMC modules or in about 70% of SUMC's total logic (or in 15 modules or 62.5% of SUMC's total logic if the changes suggested in the previous section were implenented). However no changes are required to present SUMC LSI modules in order to add these tests. Parity is encoded at the output of the ALU and is tested at the output of the floating-point MUX during all fixed point operations, and at the outputs of the SPM, MROM, and IAROM modules.

Parity checks on the IAROM, MROM, SPM and IR of Figure 22 are straight forward and will not be illustrated here. A number of exclusive-OR gates equal to the word length of these four memories and registers (152 bits) is required to perform the parity checks. All odd numbers of errors in each memory or register will be detected.

The operation of the parity encoder for the MRU of Figure 22 is described in Table XIII. The parity bit associated with the PRM is derived from an appropriate subset of the 36 bit ALU output depending on which shift function the PRM is performing. The parity bit for the MAM is normally obtained by adding modulo 2 the MAR bits included in the present MAR parity bit sum, but excluded from the new MAM parity sum by the designated shift operation, plus any new ALU bits shifted into the MAM output to the present MAR parity bit. The only exception to this is when the nonshifted ALU output is selected by the MAM. In this case the parity sum consists of the ALU output bits. The parity bit for the MQM is obtained in a similar manner to that for the MAM parity bit with the

TABLE XIII. MRU PARITY ENCODING CONTROL TABLE

PRM:	$D(N+4)_l$	$D(N+4)_s$	$D(N+2)$	$D(N+1)_l$	$D(N+1)_s$	$D(N)$	$D(N-1)_l$	$D(N-1)_a$	$D(N-4)_l$	$D(N-4)_a$
ALU_{MSB}						X	X		X	X
ALU_{MSB+1}				X	X	X	X	X	X	X
$ALU_{MSB+2,3}$			X	X	X	X	X	X	X	X
$ALU_{MSB+4...LSB-4}$										
$ALU_{LSB-3...-1}$	X	X	X	X	X	X	X	X		
ALU_{LSB}	X	X	X	X	X	X				
ALU_{LSB+1}	X			X						
$ALU_{LSB+2...4}$	X									
MAM:	$B(N+4)$	$B(N+2)$	$B(N+1)$	$B(N)$	$B(N-1)_l$	$B(N-1)_s$	$B(N-4)_l$	$B(N-4)_s$	ALU	
$MAR_{MSB+0...3}+P_{MAR}$	X	X	X	X	X	X	X	X		
$MAR_{LSB-3...-1}$							X	X		
MAR_{LSB}					X	X	X	X		
$ALU_{28...30}$							X			
ALU_{31}					X		X			
ALU_{32}				X						
ALU_{33}			X	X						
$ALU_{34,35}$		X	X	X						
P_{ALU}									X	
MQR:	$R'_3(N+1)$	$R'_3(N+1)$	$R'_3(N+4)$	PRM						
MQR_{MSB}	X	X								
MQR_{MSB+1}	X									
$MQR_{LSB-3...-0}$			X							
P_{MQR}	X	X	X							
P_{PRM}				X						

X indicates bits to be added to parity check sum. Notation is from description of MRU in MSFC document S&E-ASTR-C-005.

sum consisting either of the MQR parity bit plus excluded MQR data bits or of a transfer of the PRM parity bit depending on the selected multiplexer outputs. The parity encoder logic contains holding flip-flops for the added parity bits clocked by the same signals that clock their associated MRU registers. No access is needed to internal signals on the MRU modules so the parity encoder module can simply be added to SUMC with no changes in the existing logic. If silicon-on-sapphire CMOS logic is used and all encoder logic is placed on one module, logic propagation delays through the encoder should not be significantly greater than those through the MRU since most logic propagation delays in this case would be associated with inter-module lead capacitances rather than with individual gate delays. Total circuit complexity is 342 equivalent gates; 69 external connections are required.

ROM parity is encoded when the ROM's are designed. SPM data always comes through the MQM and MQR register and is encoded as described above. Data going to main memory is Hamming-plus-parity encoded upon entering the memory. It is then stored in the memory where it is retested prior to transmission to the IR. Hence if a parity error is discovered in checking the IR it can be attributed to sources within the processor unit with a high probability. Data in the MAR or PRR may be reused within the processor without going to the main memory unit. The most effective place to check the parity of these registers is at the FPMX module output on operations where the FPMX is not performing a floating point data normalization, (i.e., on virtually all processor microinstruction steps). Making the check at this point also tests for correct fixed-point operation of the FPMX, checking for stuck-on "1" faults of floating point gates and stuck on "0" faults of fixed point gates within the multiplexer and catching about 40% of floating point multiplexer module faults. A method for testing for the remaining FPMX failure modes, which would generally show up only during floating point instruction execution, is described in the next section but for many applications the more limited check or the elimination of the FPMX as discussed in the previous section could be the preferred alternative due to the high cost of a complete check. The logic for performing the parity check on the FPMX output requires 128 gates. This logic plus the IR and semiconductor memory parity checkers could be partitioned onto three identical LSI modules, each having dual 33 bit parity checkers and using 256 gates and 70 external connections. The 90 bit wide MROM + IAROM check would involve all three modules.

4.6.4 Enhancing SUMC Reliability Through Adding Selective Redundancy

It is possible to detect failures in much of the remaining third of the SUMC logic not covered by the circuits mentioned above. However, the complexity of the checking logic will equal or exceed the complexity of the logic being checked and hence the decision on whether or not to add portions of it should probably be made mission dependent, i.e., how reliable does SUMC have to be and for how long? What fraction of possible errors require real-time onboard detection? Will redundant processors be used as well as intra-processor redundancy? Fixed and floating point ALU modules account for approximately 20% of SUMC complexity. The most reasonable method for checking them is to duplicate them and compare their outputs since coding techniques that are invariant under both logical and arithmetic operations and that do not slow down the processor are at least as complicated to implement as the duplicate and compare method. If two exclusive-OR gates are added to each ALU module, duplicating and comparing ALU outputs simply doubles the number of SUMC ALU modules. In terms of equivalent gates this adds 2816 gates to SUMC, a 106% increase in ALU complexity. There might be a modest processor speed reduction due to the addition of an additional on-off module delay in the signal path. If the ALU's could not be changed, two new identical comparator modules would be needed having 68 external connections and 88 equivalent gates apiece; an extremely inefficient arrangement constrained by pin limitations.

A parity-based checking method for detecting all floating-point MUX errors, with the exception of those in the operand and exponent encoder sections, has been designed using 576 gates assuming partitioning onto two identical LSI modules each having 288 gates and 75 external connections. This covers 80% of FPMX failure modes - in effect protecting 356 gates over and above those protected by the basic parity check of the previous section, a 162% overhead for error detection for those gates. This is better than duplicating and comparing outputs which would require 840 extra gates, a 236% overhead, but worse than doing the FPMX function in the MRU modules as suggested previously. The operand and exponent encoders and the exponent register need to be duplicated in any case, on an additional 271 gate module including comparison logic for the exponent encoder outputs. This module replaces the MRU module presently used for the SUMC exponent register.

The FPMX parity check circuits operations are described in Figure 24 which shows which FPMX input bits will appear in the FPMX output during different shift operations. The symbol m indicates that the most significant 32 bits of the shifted input are selected, ℓ indicates that the least significant 32 bits are selected - both with single precision input selection, for double precision both the md and the m, or the ℓd and the ℓ inputs are selected. The parity check logic adds FPMX input bits not appearing in the FPMX output to FPMX output bits modulo 2 and compares this sum containing all PRR bits and/or all MAR bits depending upon the FPMX operation with the appropriate parity bits for these two registers to test for possible errors.

ALU and floating point MUX tests raise the probability of error detection given an error in SUMC to about 95% but requires 4 times as much error correction logic as detecting 70% of the failure modes does. The remaining SUMC logic performs control functions and is of a very random nature and hence difficult to test efficiently for errors. A brute force approach where all SCU, FCU and DCU module functions are duplicated and compared at module outputs would require 96 comparisons to be made and represents an upper bound on the complexity for checking these modules. This approach would probably involve redesigning the 3 modules to include comparisons between duplicated modules as in the case of the ALUs since use of external comparator gates would quickly run into pin limitations. The overhead for fault checking could average 157% for this logic. If only a partial check were performed the cost could be reduced. For example, the overhead for checking the SCU module is 125%.

4.6.5 Summary and Recommendations

Proposed modifications to the SUMC design to adapt it to ARMMS requirements include: 1) incorporation of voter/switch and replicated memory bus interfaces to allow processor operation in simplex, duplex, and TMR modes with ARMMS memories; 2) addition of parity check networks to detect faults in internal

ER$_{S}$, 4, 5, 6, 7 =	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111
SHIFT OPR =	N+64	N+60	N+56	N+52	N+48	N+44	N+40	N+36	N+32	N+28	N+24	N+20	N+16	N+12	N+8	N+4	N	N-4	N-8	N-12	N-16	N-20	N-24	N-28	N-32	N-36	N-40	N-44	N-48	N-52	N-56	N-60
PCI CONTROLS = FPM msb + 0, 3																	m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ
PC2 FPM msb + 4, 7																m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	
PC3 RPM msb +8, 11															m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ		
PC4 FPM msb + 12, 15														m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ			
PC5 FPM msb + 16, 19													m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ				
PC6 FPM msb + 20, 23												m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ					
PC7 FPM msb + 24, 27											m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ						
PC8 FPM msb + 28, 31										m	m	m	m	m	m	m	m	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ							
PC9 FPM msb + 32, 35									md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd								
PC10 FPM msb + 36, 39								md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd									
PC11 FPM msb + 40, 43							md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd										
PC12 FPM msb + 44, 47						md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd											
PC13 FPM msb + 48, 51					md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd												
PC14 FPM msb + 52, 55				md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd													
PC15 FPM msb + 56, 59			md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd														
PC16 FPM msb + 60, 63		md	md	md	md	md	md	md	md	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd	ℓd															

SEE TEXT FOR NOTATION EXPLANATION

32410-40

Figure 24. FPM Parity Check Control Table

memories, and in most registers and multiplexers; 3) control of timing from a central clock to assure synchronism during duplex and TMR operation; 4) additions of BOSS interfaces for assignment control and power switching; 5) MROM and logic modifications as necessary to enhance processors speed and reliability and minimize complexity.

Table XIV breaks down SUMC complexity by functional blocks and lists the cost of fault detection to each block in number of gates for both the baseline SUMC and the simplified version of SUMC suggested in the second section of this report. Figure 25 shows the relationship between increasing fault detection coverage and adding redundancy for these two versions of SUMC.

The ARMMS CPE baseline's coverage vs complexity is also shown for comparison. Note that the complexity reduction measures discussed in this report allows a 50% reduction in SUMC failure rate over a wide range of coverage trade-offs, when compared to a baseline SUMC with an equivalent amount of added redundancy.

TABLE XIV. SUMC COMPLEXITY BREAKDOWN

	Baseline SUMC			Simplified SUMC		
Module	No. of Gates		Fault Detection Gates	No. of Gates		Fault Detection Gates
MROM*	4500			750		
IAROM*	500			0		
IR	200	(9260)	1078	200	(3654)	1078
SPM*	2000			1000		
MRU	1704			1704		
FPMX**	356			0		
ALU/EALU	2640		2816	1320		1496
FPMX/ER**	564		804	208		228
SCU/IC	255		319	255		319
CLT	417		737	417		737
Total	13,136		5754	5854		3858

*4096 Bit ROM and 256 Bit SPM modules estimated equal in complexity to a gate chip having 250 gates.
**See text for breakdown of assumed FPMX failure modes into two categories.

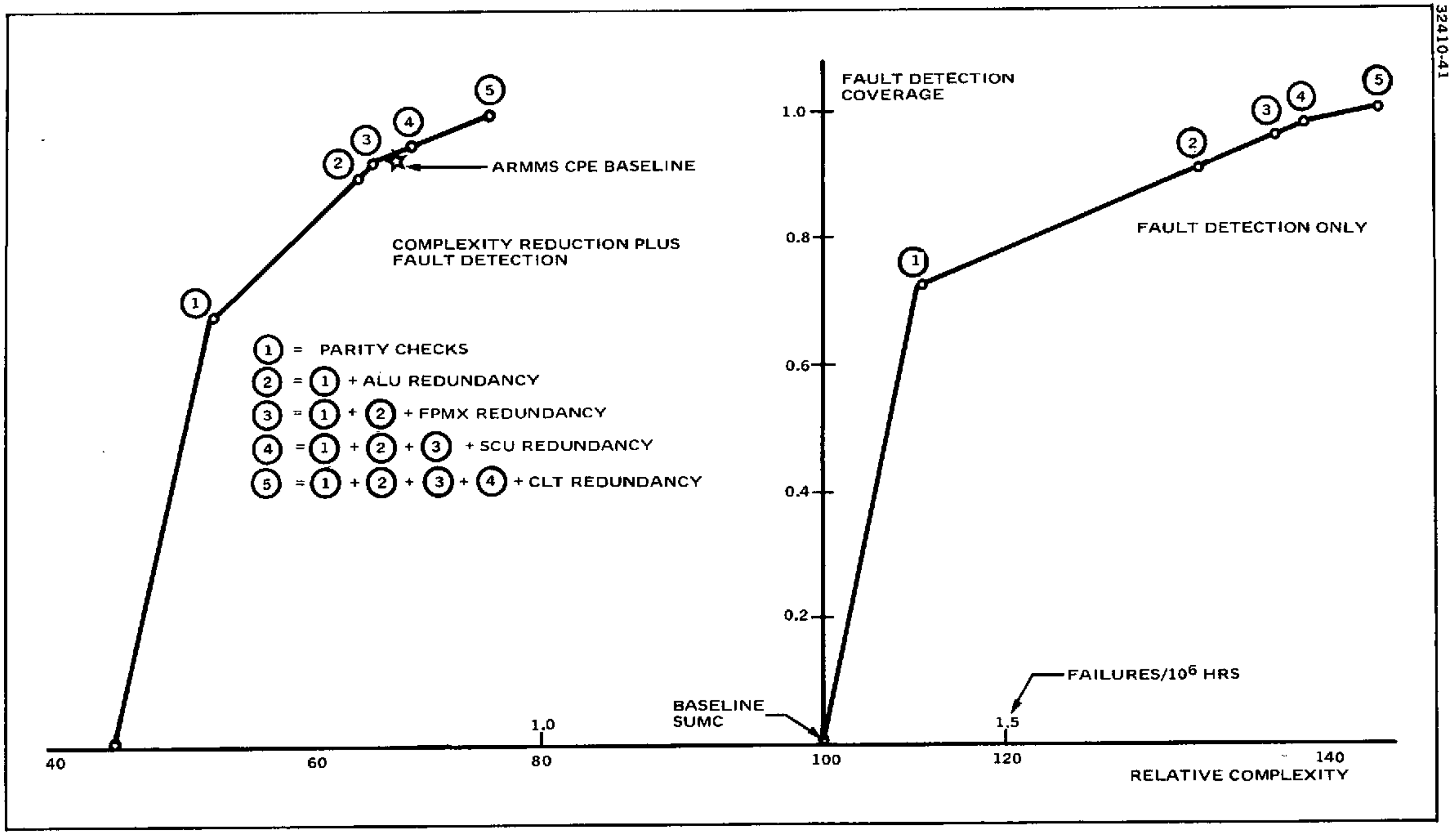


Figure 25. SUMC Fault Detection Summary

Note also that while an ARMMS CPE based on SUMC building blocks can achieve reliability improvement comparable to the ARMMS baseline only through the redesign of some SUMC modules, a SUMC processor can be made significantly more reliable with minimum redesign. The amount of redesign required would depend on the stringency of a particular mission's requirements. All of the alternatives listed require substantially less complexity than that for duplication of complete processors. The failure rates shown in the diagram assume the ARMMS gate failure rate of 10^{-10} failures/hr/gate for late 1970's CMOS SOS LSI logic. New LSI modules recommended for addition to the basic SUMC LSI module set for reliability enhancement are listed below. It appears that a majority of SUMC failure modes can be detected and ARMMS reliability enhancement techniques applied while using the basic SUMC LSI module set plus these additional modules. However serious consideration should be given to simplifying SUMC ALU, FPMX, SPM and ROM modules if failure modes are to be minimized.

ADDITIONAL SUMC MODULE RECOMMENDATIONS

No. Needed	Description	Pins	Gates
3	Parity Checker	70	256
1	MRU Parity Encoder	69	342
1	Hamming Encoder	40	224

4.7 A BOSS-Less Version of the ARMMS Computer

A major objective of the ARMMS Computer study has been to achieve a modular design which allows for a family of highly reliable computers in a wide range of configurations suitable to a wide range of space missions. It is expected that some missions requiring ARMMS reliability will not require the high computational capacity provided by ARMMS multiprocessing and that a simplified version of ARMMS without multiprocessing would be a desirable member of the ARMMS family of computers. This report describes the system design of such a computer.

In ARMMS, executive functions including program dispatching, interrupt handling, and reconfiguration control are centralized in the BOSS module which is operated in the TMR mode for maximum reliability. In addition BOSS has non-processing functions such as power and timing control and distribution. If either the multiprocessing or the reconfiguration (simplex, duplex, and TMR) requirement were dropped from ARMMS the BOSS processing functions could be handled by the CPE modules although the non-processing functions would still need to be centralized. The Hughes H-4400 computer is an example of such a simplex multiprocessor.

In the full version of ARMMS BOSS'es dispatching of programs to the various processors requires dynamically varying the assignments of each physical processor module between simplex, duplex, and TMR modes as a function of program execution requirements. This takes up a considerable portion of BOSS time but is practical with a BOSS processor in the system. However if this job were done by the CPEs in a duplex or TMR configuration computation would be slowed significantly and if it were done by a simplex CPE with error detection coverage less than unity undetected erroneous operations might result compromising ARMMS reliability objectives. However if all simplex programs are actually assigned to simplex streams, all TMR programs to TMR streams, etc. reconfiguration is decoupled from the dispatching problem and multicomputing is possible without either the dispatching inefficiencies or the potential reliability degradation noted above and without the requirement for a BOSS processing capability. Program dispatching and external and I/O interrupt handling are distributed among the CPEs. Fault interrupts and reconfiguration around failed modules are handled by hardwired logic added to the other non-processing functions retained from the simplified BOSS module. The resulting module is called mini-BOSS and is expected to have no more than 20-25% of the complexity of a BOSS module containing a processor.

If ARMMS IOPs are connected one-to-one with CPEs, as in the ARMMS full-processing stream concept, no processing capability would have to be included in IOPs of a non-multiprocessing version of ARMMS simplifying these modules by 50-60%. This capability was originally included to reduce the processing load on BOSS while retaining a centralized I/O processing capability.

So far the only change to ARMMS capabilities by eliminating BOSS and I/O module processing is to change the second "M" in ARMMS from "multiprocessor" to "multicomputer." It is instructive to see what multicomputing costs as an option and consider a mission dependent choice between a multicomputing BOSS-less ARMMS and a BOSS-less ARMMS having a single reconfigurable stream. Principally the single stream reconfigurable computer requires only an active/inactive indication from mini-BOSS to each module instead of a stream assignment code saving approximately 4% in overall system logic complexity by reducing mini-BOSS storage and control lines to and assignment decoders in other modules.

If a global memory capability is required for multicomputing so that the streams can talk to one another, memory access control logic very similar to that for a full ARMMS configuration is required adding approximately 2% to the overall system logic. In addition the processors must include an instruction similar to the TEST AND SET found in IBM system 360, and 370 computers to provide for global memory access control since with no BOSS processor dynamic access control by means of base and bounds registers is not possible. It should be noted that global memories while convenient present a potential reliability hazard in that the access control method proposed only works if it is used - i. e. there is no protection involved if a program accesses a restricted location in memory either willfully or due to an undetected malfunction in the simplex mode. Memory protection using a lock and key approach could be employed but then a simplex processor could restrict access to the wrong set of locations due to a malfunction. Since the cost in complexity and the probability of a malfunction due to multicomputing with global memory access are both small this might prove to be a useful option for many missions but it is not regarded as a required characteristic in a minimal ARMMS computer since there might be no requirements for one stream to communicate with another and if there were communications could take place through the IOP's rather than through the memories if necessary.

4.7.1 System Level Changes for a Boss-Less ARMMS

Aside from the optional status of global memories there are two other system level changes for a BOSS-less ARMMS. First while status and control communication between BOSS and other modules was via a BOSS to Module Bus (BMB), with each module containing BMB interface logic and a status register, all system state storage is retained in mini-BOSS and communicated to the various modules via levels on discrete control lines. This is principally due to the fact that while BOSS fetched system state information from main memory and then relayed it to other modules mini-BOSS stores such information internally in redundant power protected CMOS registers.

A second consideration is that since mini-BOSS hardwired logic for replacing faulty modules is much more constrained than that of BOSS, mini-BOSS will not be able to determine which of two processors or memories are at fault in the duplex mode in cases where the fault is detected by the voter switch but not by logic within the faulty module. This results in moving the Hamming error detection/correction logic that was placed in the processor modules in earlier versions of ARMMS to the memory module outputs instead. While the old location provides for slightly increased masking of processor errors and for reduced hardware in systems where the number of memory modules exceed the number of processors it does complicate the error detection process. It should be noted that with mini-BOSS the three ARMMS operating modes provide the following fault detection and masking capabilities:

{% Detect/Mask}

Module	Simplex	Duplex	TMR
CPE	93/0	99+/93	99+/99+
IOP	59/0	99+/59	99+/99+
Memory	99+/70	99+/99+	99+/99+

Most faults are detected in simplex but only a portion of those in the memory are masked. Duplex operation guarantees that virtually all faults will be detected avoiding erroneous computations but only those faults also detectable in simplex can result in masking and replacement of faulty modules with spares. The masking property means that the computer is able to complete programs already in progress before switching in a spare just as in the TMR case and that it can continue to operate in the presence of a maskable fault once available spares have been exhausted until ARMMS is commanded to change to a configuration requiring fewer active modules. Finally TMR operation masks virtually all errors through voting. Clearly all modes have distinct characteristics which distinguish them from one another except in the special case where all modules internal error detection coverage approaches unity making duplex operation equivalent to TMR operation in performance. As discussed in earlier reports unity coverage in the processor modules results in excessive complexity for these modules in the ARMMS context and in incompatibility with existing SUMC logic and is not recommended.

Aside from simplification of BOSS and memory access control logic in the CPE; IOP, and memory modules and the elimination of the requirement for processing other than for the channel within the IOP the only other change required outside of mini-BOSS is the addition of interrupt and timer logic within each CPE to handle I/O and external interrupts since the functions are no longer handled by BOSS. Actually this brings the CPE closer to the SUMC design since SUMC had to handle its own interrupts.

4.7.2 Miniboss Concepts

As mentioned earlier mini-BOSS retains BOSS power and timing distribution functions and co-ordinates ARMMS reconfiguration processes, either due to new assignments from outside commands or due to detected malfunctions in other ARMMS modules. Mini-BOSS is made TMR redundant with all partitions powered and all outputs voted. For a 5 year mission and an assumed mini-BOSS complexity of under 2,000 gates per partition (about 25% of the number required by a BOSS partition "A") failure rate calculations show a 99.98% chance of no non-maskable failures and a 97.4% chance of no failures whatsoever for mini-BOSS logic without requiring additional switchable spares.

Mini-BOSS keeps track of the status of each module and of its stream assignment if the multicomputer option is included. A module can take on one of four states: spare, active normal, active rollback, failed. Initially all modules are spares. A ground command places some subset of the available modules in the "active normal" state and gives them assignments as discussed below. If a module fails and the failure is detected mini-BOSS receives the failure interrupt immediately if the failure was unmaskable or at the end of the program segment if it was maskable and places that module in the "active rollback" state and requests the module to repeat that program if the failure was non-maskable or to proceed to the next program if the failure was maskable. If that module completes the assigned program successfully it will be returned to the "active normal" state, if it does not it will be placed in the "failed" state and its assignment will be transferred to the first available spare module. The program to be executed is determined by software - i.e. whether mini-BOSS receives the fault interrupt before or after the program status block is updated. If the block has been updated the next program is executed, if it has not the present program is repeated. Program logic is expected to be constructed in such a way that it can be repeated if necessary. The Program status block containing the contents of all important processor scratchpad memory registers is stored in a unique block of locations in main memory for each processing stream.

If only one stream is involved mini-BOSS need only tell a processor module whether or not it is active. The number of active processors then determines whether the stream operation is simplex, duplex, or TMR mode. If more than one stream is involved, however, each active module must be given an assignment code uniquely specifying its stream assignment at that point in time. A three bit assignment for each of 4 CPE's could allow each the following possibilities: {TMR, DUPLEX A OR B, SIMPLEX A, B, C, OR D, SPARE} allowing for all possible combinations of 4 processors. Memories can either be given an explicate stream assignment (in a system with no global memories) or can include access control logic capable of responding to any stream's access request as in the case of the full ARMMS system. In addition each memory must receive a page assignment containing the most significant bits of that memory's address and implicate information as to the proper output bus to respond on in the case of accesses by duplex or TMR streams where more than one memory module is given a redundant page assignment. This was discussed at length in the ARMMS Phase II report. In addition to the page assignment which is communicated to the memory by mini-BOSS each memory page is labeled "essential/ non-essential" internal to mini-BOSS. An essential memory would contain programs and important tables the loss of which could disable a stream. Upon a failure only essential memories would be reloaded from the remaining good memories in duplex or TMR modes and loss of one of these memories would halt operation in simplex. Loss of a non-essential memory would be handled by a replacement procedure identical to that for the loss of a processor. Reloading of an essential memory requires first clearing the newly activated memory's contents by an interrupt from mini-BOSS to the memory and then causing the data in the good memories to be READ out and then read back into both the good and the newly activated memories by a special "RELOAD memory" routine activated by an interrupt from mini-BOSS to the CPE.

Since one stream may use more than one memory page it is necessary for a memory which has internally detected a failure to communicate this fact to that stream's CPE(s) or to the last CPE to use a global memory. If a CPE receives a failure indication from a memory it stores the memory page address in a reserved location in its internal scratchpad memory and sets a control flip-flop. If the memory failure is maskable operation continues until the program is complete at which time mini-BOSS receives a failure interrupt or, if it is not maskable, operation ceases immediately and mini-BOSS is interrupted. In each case mini-BOSS receives interrupts from both its CPE and the memory and once the memory has been replaced and the new memory's contents cleared if necessary the CPE is restarted by mini-BOSS and told either to reload an essential memory and then resume computations according to information stored in that module's program status block or to simply resume computations without reloading a non-essential memory.

For 4 CPE, 4 IOP and 8 main memory modules 112 bits of internal storage would be required within each mini-BOSS partition to implement the functions discussed above. Of these bits, 72 would control lines to other modules and 40 would be use only internally by mini-BOSS. In addition 20 command lines, 3 clock sync lines and 17 power lines would be required for a total of 112 lines from mini-BOSS to other modules. If each processor was provided with 2 status interrupt lines to communicate the states {operating, memory failure, processor failure, program completed successfully} to BOSS and each memory was provided with a single status line a total of 24 additional lines to mini-BOSS would be required giving a total of 136 lines at the mini-BOSS interface.

4.7.3 Conclusions

Due to its simplicity relative to a full ARMMS system, a "BOSS"-less version of ARMMS will probably be the version implemented in the ARMMS breadboard. In a real-time environment many programs will be of a repetitive nature and it may be possible to achieve throughput equal or greater to that obtainable with a multiprocessor for a multicomputer since the programs can be distributed equally among the available streams based on simulations of ARMMS on a ground-based computer prior to a flight and the distribution will be subject to less of the randomness associated with multiprocessing. If a mission is found for ARMMS so that specific program requirements could be defined it could be informative to simulate both multiprocessing and multicomputing and compare their throughputs vs. their relative complexities to see if a full ARMMS system is justified or if the system just described is superior.

4.8 Requirements of the Automatically Reconfigurable Modular System

The objective of the Automatically Reconfigurable Modular System (ARMS) project is the detailed design, fabrication, and testing of an ARMS breadboard to prove the concepts developed to date in the ARMMS study. The breadboard's processor will utilize the 32-bit breadboard version of MSFC's SUMC processor as a baseline with modifications where necessary to meet ARMS requirements. ARMS communication with SUMC's LSI module set and with SUMC's instruction set, which is a subset of IBM's system 360 instruction set, should minimize costs associated with software and LSI development, both for the ARMS breadboard, and in application of ARMS to any potential future missions. Schedules, deliverable items, and costs will be so divided that this project can be incrementally funded on an annual basis: First a simplex breadboard will be developed; then the breadboard will be fabricated and tested; third, memory and processor modules will be replicated so that single processing stream duplex and TMR configurations with switchable spares can be demonstrated; finally options such as multiple stream operation, multiprocessing, LSI module fabrication, and/or complete LSI breadboard construction and testing can be undertaken if these meet NASA interests and requirements later in this project.

Concepts which ARMS will be required to verify and demonstrate include: 1) variable configuration capability ranging from fully synchronous TMR operation to maximize reliability to simplex operation for longer life in the presence of failed modules, and for highest throughput in the event that it is later chosen to implement a computer capable of supporting more than one processing stream; 2) high reliability through the incorporation of fault detection and recovery features such as error detection and correction codes, selective redundancy, switchable spare modules, voting and comparison techniques, and the use of high reliability components; 3) modular design to provide a family of computers responsive to many mission types and phases.

An ARMS breadboard based on these criteria will have the following required specifications:

1. The total system shall incorporate 4 Central Processing Elements (CPE) utilizing SUMC architecture, 4 main memory modules, one Input/Output Processor (IOP), 4 memory to processor buses, 4 processor to memory buses, a central (configuration) control "Mini-BOSS" element and sufficient peripheral equipment to exercise the system. All CPE, IOP, and memory modules must incorporate voter/switches at their inputs. This is the minimum configuration capable of demonstrating voting, standby sparing, and synchronization;

2. The system shall be capable of simplex, duplex, and TMR operation with switchable spares. It shall be required to support only one processing stream although support of multiple processing streams may be considered on an optional basis;

3. Fault insertion and breadboard control and monitoring capabilities shall be provided;

4. System software to exercise the breadboard in order to demonstrate fault detection and/or masking, recovery and system throughput shall be developed in such a way as to be compatible with existing available support software such as assemblers, compilers, loaders, link editors, etc. Only software unique to the ARMS breadboard will be developed by this program;

5. ARMS shall incorporate the necessary logic for single error correction and multiple error detection within memory modules by means of a Hamming code. The CPE and IOP shall contain error detection logic internally where practical. The central (configuration) control element will provide timing and synchronization signal generation and distribution, power sequencing, and minimum hardwired reconfiguration and self-test capabilities based on error detection inputs from other ARMS modules;

6. The ARMS IOP will be capable of interfacing with MSFC's Data Management System Breadboard (DMS) and with ARMS peripherals consisting of a printer, a keyboard, and either paper or magnetic cassette tape storage. Both the DMS and ARMS peripherals shall be capable of being connected to the IOP simultaneously through separate selector type channels.

7. Logic functions shall be designed to be compatible with the SUMC LSI module set where practical and so as to simplify transition into new LSI modules in the case of ARMS functions not presently included in the SUMC LSI module set.

8. Documentation adequate to allow understanding, operation, and troubleshooting of the system hardware and software will be provided. This will include flow charts, program and wire listings, operating instructions, principles of operation, logic diagram and mechanical drawings.

Designs that have been developed under the existing ARMMS project will be used where possible or expanded or modified where necessary to implement the ARMS breadboard. A major objective of ARMMS has been to achieve a modular design which allows for a family of highly reliable computers in a wide range of configurations suitable to a wide range of space missions. It is expected that some missions requiring ARMMS reliability will not require the high computational capacity provided by ARMMS multiprocessing and that simplified version of ARMMS without multiprocessing would be a desirable member of the ARMMS family of computers. Such a system provides a reasonable lowest cost baseline for the ARMS breadboard and comes closest to meeting the real requirements for potential space missions to which ARMS could be applicable since while, these missions require very long lived computer none have to date demonstrated a need or desire for multiple processing streams. Expected differences between ARMMS and ARMS are summarized in Table XV.

TABLE XV. DEVIATIONS FROM ARMMS FOR ARMS

- System Level Changes
  - Single Processing Stream Rather than Multiprocessing
  - No Global Memory
- CPE Changes
  - BOSS Control Interface Simplified for Central Control Element
  - Memory Access Control Logic Modified
  - Interrupt Logic Added
  - More Compatible with SUMC LSI Modules and Instruction Set
- Memory Changes
  - Access Control Logic Simplified
  - Half Word Addressing Allowed
  - Error Detection and Correction Logic Moved from Processor to Memory
- IOP Changes
  - No Processing Required Other than for Channel — Complexity Reduction ~50-60%
  - BOSS and Memory Interfaces Modified as in CPE
  - IOPs are Paried with CPEs by Central Control Element
  - Second Channel Provided for TTY Interface to Peripherals
- BOSS (Central Control Element) Changes
  - No Processing — Complexity Reduction ~80%

SECTION 5

ARMMS COMPONENT AND PACKAGING TECHNOLOGY STUDIES

This section consists of two parts. The first summarizes the component technology tradeoff studies performed during Phases I and II in the areas of data bus technology, logic families, and power supply configurations. CMOS is the recommended choice for ARMMS basic logic because of its low power dissipation, and high noise immunity and packaging density. Other CMOS advantages include wide temperature operations, high fanout, easy interfacing with bipolar circuits, and operation over a wide power supply voltage range.

Bus technology studies resulted in the choice of a current source drawer operating into a single ended isolated receiver over a 50Ω microstrip line to provide best power-speed characteristics with simple technology and minimum pin count.

Power supply configurations ranging from a single centralized supply to individual power supplies per module were considered. Since no module should depend on one power supply, modularization must be effective over a range of ARMMS configurations, and BOSS must be able to switch other module's power on and off, a partially centralized regulator supplying power to up to 5 modules, each of which incorporates a simple DC/DC converter, was selected as the best alternative.

The last portion of this section gives the results of a study to define packaging concepts and physical hardware parameters for each of the ARMMS module types and for a range of typical ARMMS configurations. Areas investigated included LSI chip and discrete component packaging methods, printed circuit board design, chassis design, module interconnection techniques, and thermal and stress analysis of the design chosen. For configurations ranging from 4 through 37 total modules the volume ranged from 945 in.^{3} (15,500 cm^3) to 5600 in.^{3} (91,900 cm^3), weight (mass) ranged from 33 pounds (72.6 kg) to 194 pounds (426.8 kg) and power ranged from 120 watts to 1825 watts.

[illegible]

5.1 ARMMS COMPONENT TECHNOLOGY STUDIES SUMMARY

Component technology studies were performed for the areas of data bus technology, logic families, and power supply configurations the results of which were described in detail in earlier phase reports and are summarized in this topic.

5.1.1 Data Bus Study

ARMMS data bus transmission line and interface logic designs should allow 10 MHz data transmission between modules in any ARMMS configuration with an average bus power dissipation of 250 mw/bit. To reduce pin counts single ended (rather than differential) current source receivers and drivers will be used, transmission power is minimized as much as possible without degrading data transmission quality, and a synchronous clock system with a period greater than worst case delays in bus and module interfaces is required to allow lock-step operations in duplex and TMR modes.

Since data is bussed to many modules it is important that a failed module not be able to short the signal bus. Then, if a module fails open, while the module is not available for use the bus is still available to the rest of the system. To isolate receivers resistor isolation may be sufficient. For most driver schemes a switch is necessary. For current drive transmission it is necessary only to provide a switch in series with the high state supply for the drivers to isolate a failed or unused module from the signal bus because these drivers have a low impedance path from the signal bus to the +5.0V supply, but no low impedance path from the signal bus to ground. In the ARMMS driver no one component failure can disable a signal bus.

5.1.2 Logic Family Study

Four logic families were investigated for use in ARMMS: Standard TTL, Schottky TTL, Low-power Schottky TTL, and CMOS. The first two families must be eliminated due to heat dissipation problems in ARMMS limited processor volume. A Schottky TTL implemented processor module would require a structure thickness of one inch. Even if more exotic cooling systems such as heat pipes were employed managing the power densities of these families would be a nearly impossible task.

Without question the speed and propagation delays of today's low power Schottky TTL are adequate for ARMMS. Present day CMOS devices, however, have propagation delays several times those of the low power Schottky family but their future looks bright. Semiconductor houses are today developing ion implantation techniques and silicon gate technologies for CMOS to reduce parasitic and junction capacitances. Silicon on sapphire and silicon on spinal substrates will dramatically reduce substrate capacitance associated with bulk silicon CMOS devices. Device speeds in the range of 100 MHz will be possible without effecting the speed-power relationship established in present day hardware. However off chip capacitance considerations limit chip to chip logic speeds to approximately 40 nsec.

DTL and RTL logic lines were rejected because it is thought that their future in new design is somewhat limited because of their relatively low speed and lack of interest within the semiconductor houses themselves. PMOS is definitely a possible choice, but has been rejected because of the superior speed, lower power dissipation and greater interest in CMOS. The high power levels of ECL combined with the fact that their speed is not needed has caused the elimination of this logic line from consideration.

Low power Schottky TTL's principal advantages are its high speed and long history of reliability. Its disadvantages are its requirements for a well regulated power supply and the need for careful layout and many "anti-glitch" bypass capacitors.

At speeds of approximately 5 MHz, gate dissipations should run approximately 3 mw rivaling that of CMOS. Arrays of 60 gates are already available and prospects for 80 gate arrays for custom chips still seems good. Asumming a device complexity of 80 gates, approximately 200 chips would be necessary to complete a processor module. The large number of devices required, and the interconnections necessary, tend to rule out the use of this logic family in the ARMMS computer. Aside from the interconnection problems, the low power Schottky T^2L element would be an ideal device for systems use. It does not seem likely, however, that arrays in excess of 250 gates will become available in the near future. Nor does it appear likely that processing yields will allow anything other than discretionary wiring techniques for reaching this level of complexity.

The advent of CMOS digital elements has given system designs a new feel which solves many of the problems of bipolar hardware. One of the most desirable characteristics of CMOS is its lower power dissipation. Under quiescent conditions either the p channel or n channel device is off; consequently, the device is dissipating virtually no power. Only during the transition between states does the device dissipate power. Quiescent power dissipation is typically 0.01 μw per gate; dynamic power dissipation is 0.4 mw/MHz. With a lightly loaded (6 pf) line, a CMOS gate will dissipate approximately 2 mw at 5 megahertz.

The ability to operate CMOS from a single, relatively wide tolerance supply bus significantly eases system power supply design requirements. Most CMOS logic is fully capable of working from a supply voltage from as low as 4 volts, to as high as 18 volts. Noise immunity of CMOS elements is correspondingly high. Noise immunity is typically 0.45 Vdd and increases with increasing supply voltage.

Another significant advantage of CMOS is simplicity of fabrication. CMOS requires three major diffusion steps compared to five for bipolar devices. Device geometries for CMOS are significantly smaller than for bipolar elements, and linear resistors are not used. Consequently, a CMOS gate may be as much as a factor of eight smaller than its bipolar counterpart. This, combined with simpler fabrication processes, will allow high complexity chips with moderate yields. Also CMOS elements are capable of operating over the entire military temperature (-55 to +125°C) with only minimum variations in device performance and because of relatively low output impedance and high input impedance, CMOS has the largest fanout capability of any logic form. Fanouts of greater than 50 are readily achieved. Interfacing with bipolar logic elements is also relatively simple. CMOS will interface with T^2L directly, and open collector T^2L will directly interface with CMOS. A pull up resistor is normally required to interface T^2L elements to CMOS inputs.

If a processor module was to be constructed today, and if the processor had to operate at speeds in excess of 5 MHz, low power Schottky T^2L would be selected as the best logic choice. CMOS would have to be rejected because of its somewhat lower speed. This would be the only reason for its rejection.

Since the design of this computer is being projected into a future time frame, CMOS processing should have proceeded to the point that its speed characteristics will equal or surpass that of low power Schottky T^2L. With the higher speed, lower power, greater fanout, ease of T^2L interfacing and greater allowable device complexity, it is recommended that the CMOS logic elements be chosen as the basic logic elements in the ARMMS computer.

5.1.3 Power Supply Configuration Study

A power supply configuration tradeoff study was performed during Phase II to select a power distribution implementation and to generate detailed circuits allowing the determination of parts counts, weight, power, and reliability of the baseline design.

Three bus systems were investigated for primary power distribution: a regulated AC bus, a low voltage DC bus, and a conventional DC bus. The latter alternative was chosen because it is a proven design providing module ground isolation, minimal noise problems, and ease of power switching. Its only disadvantage is its relatively high parts count.

Six secondary power distribution methods were considered ranging from a single supply with redundant backup providing power to all modules to a separate supply for each module. A "partially decentralized" approach was chosen in which each module has its own power supply operated from a regulated dc bus supplied by a common pre-regulator. Each pre-regulator supplies several modules with the number of pre-regulators depending on the size of the total system configuration. No regulating circuitry is required in the individual module supplies saving a significant number of parts. This approach yields excellent ground isolation and regulation and good configuration flexibility at a moderate complexity. It has three other advantages: 1) good system reliability since failure of one supply will only cause one module to fail; 2) good thermal characteristics since regulator power losses are not added to other module heat sources; and 3) good output voltage flexibility since power is switched at the primary side of a DC/DC converter and thus new secondary voltages can be easily added. The distribution scheme is shown in Figure 1.

For the detailed design described in the ARMMS Phase II report power supply complexity averages less than 40 components and reliability (exclusive of fuses) in 0.066 failures/10^6 hours. The efficiency of the module power supplies is expected to run 80% and that of the pre-regulator 85%. Overall efficiency is thus 68%.

5.2 ARMMS PACKAGING CONCEPTS STUDY

The primary objective of the ARMMS packaging study performed during Phase III was to define the packaging concepts for each of the five module types: Central Processing Element (CPE), Input/Output Processor (IOP), Block Organizer and System Scheduler (BOSS), Main Memory, and Preregulator Modules. It was also necessary to investigate the weight and volume requirements of each module type and determine the ability of each module to meet the system operational requirements. In addition, a thermal analysis of key functions of the proposed design was undertaken and results of this analysis were used to reconfigure those expected problem areas.

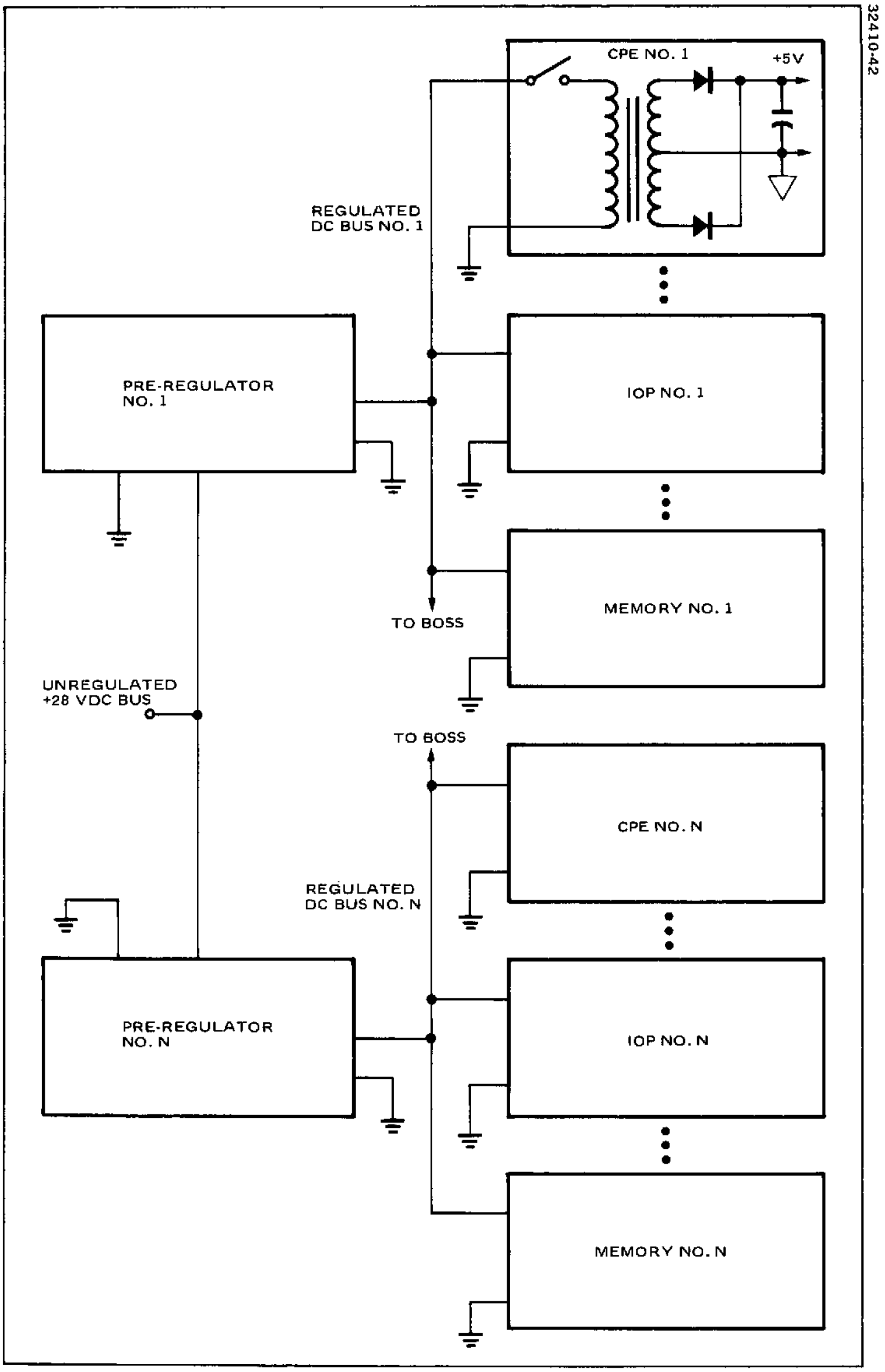


Figure 1. Typical ARMMS Power Supply Configuration

5.2.1 LSI Chip and Discrete Device Packaging

Because of the complex nature of the LSI chips to be developed for this program (>250 equivalent gates of random logic per chip), a tradeoff study was necessary to determine the optimum configuration for packaging the individual chips. Three design configurations were considered:

1. A hybrid design where 10 to 20 bare chips would be interconnected in one package.
2. A design that would package the bare chips in individual leadless packages.
3. A design utilizing beam leaded devices.

Assuming an 80% yield for die attach and wire bonding, a 10-device hybrid would have a yield of 10.7% while a 20 device hybrid would have a yield of less than one percent. A hybrid with less than 10 chips was not considered when preliminary investigations indicated only minor weight and volume improvement over the discrete package approach. The individual leadless package was adopted because it offered a high yield for device assembly (80%) plus the ability to completely environmentally test and power age the device before committing it to hardware. With a hybrid design, complete environmental testing is possible only after completion of the entire package. Any dropouts due to burn-in, assembly defects or electrical overstress would be extremely costly at that point.

The use of beam lead technology was rejected for two basic reasons:

1. The long term reliability of large (60 lead) beam leaded devices has not been proven nor have the production processing problems been fully resolved.
2. The beam lead attachment technique represents a nonoptimum thermal control design. The power dissipation levels of the chip will not allow the added thermal impedance expected with beam leads. Since power dissipated on the chip must be conducted through the beam lead to the package to provide conductive cooling, combined with the relatively long thermal path and small cross section area, local hot spots on the device could become the determining factor in limiting the operational temperature of the computer.

The interconnection of the discrete devices (resistors, capacitors, diodes, transistors, etc.) will be accomplished by conventional hybrid assembly techniques. Because relatively high values of resistors are expected with only moderate requirements on thermal coefficient of resistance, thick film processes will be used. Hybrids with 40 to 50 elements can be fabricated with yields high enough (70%) to make their use economical. Areas where the hybrids appear most attractive are those of the bus interface and the IOP buffer circuitry. Where high power dissipating circuits are found, discrete devices will be used.

5.2.2 Printed Circuit Board Design

Interconnection of the discrete LSI devices and hybrids within a module will be accomplished with a printed circuit board sandwich assembly (Figure 2). The electrically conductive patterns are stacked by planes, separated and insulated by layers of a high alumina ceramic material. The composite hermetically seals all internal circuitry. The two insulating planes will be 0.008 thick 95% Al_2O_3 ceramic. The ARMMS circuit cards are essentially two cards bonded together with a center layer of 0.010 copper for thermal conduction. Either tungsten or molybdenum based metallizing may be used for the conductive patterns. Thicknesses of 0.001 inch are normal. Conductor widths of 0.005 inch with 0.010 inch spacing is recommended, however 0.004 mil lines on 0.008 centers are feasible. Ten mil diameter holes in the insulating plane (called via holes) are used to interconnect the conductive planes. All areas which are exposed normally receive nickel and gold plating. The 30-pin (Figure 3) and 60-pin CMOS leadless packages may be attached using ultra-sonic or thermal compression bonding techniques. An alternate method is to have packages with short stub leads for bonding. Since the package substrate and the circuit card are of the same material, the common failures from thermal cycling will not be present.

The CPE circuit card (one side of the bonded assembly) was chosen as a typical example for calculating the number of layers required for the cards. The processor card has three 60-lead flat packs, three 30-lead flat packs and five 22-lead hybrid flat packs. All input/output signals go through the hybrid circuits. Therefore, 45 conductors will enter the board and 45 leads will continue out to the logic flat packs (5 x 22 = 110 minus 20 power and ground leads = 90). The size of the card is 4.312 x 3.438 inches.

Assuming:

0.25 each side for attachment

0.20 for flexible connection area

0.30 for intraconnection connector

This gives an active conductor area of:

$$(4.312 - 0.50) \times (3.438 - 0.50) = 3.812 \times 2.938 = 11.20 \text{ in.}^2$$

With conductor widths of 0.005 and 0.010 spacing and using an efficiency of 70%, the maximum possible number of vertical runs will be

$$\frac{3.812 \times 0.70}{0.015} = 177.$$

The maximum possible horizontal runs will be

$$\frac{2.938 \times 0.70}{0.015} = 136.$$

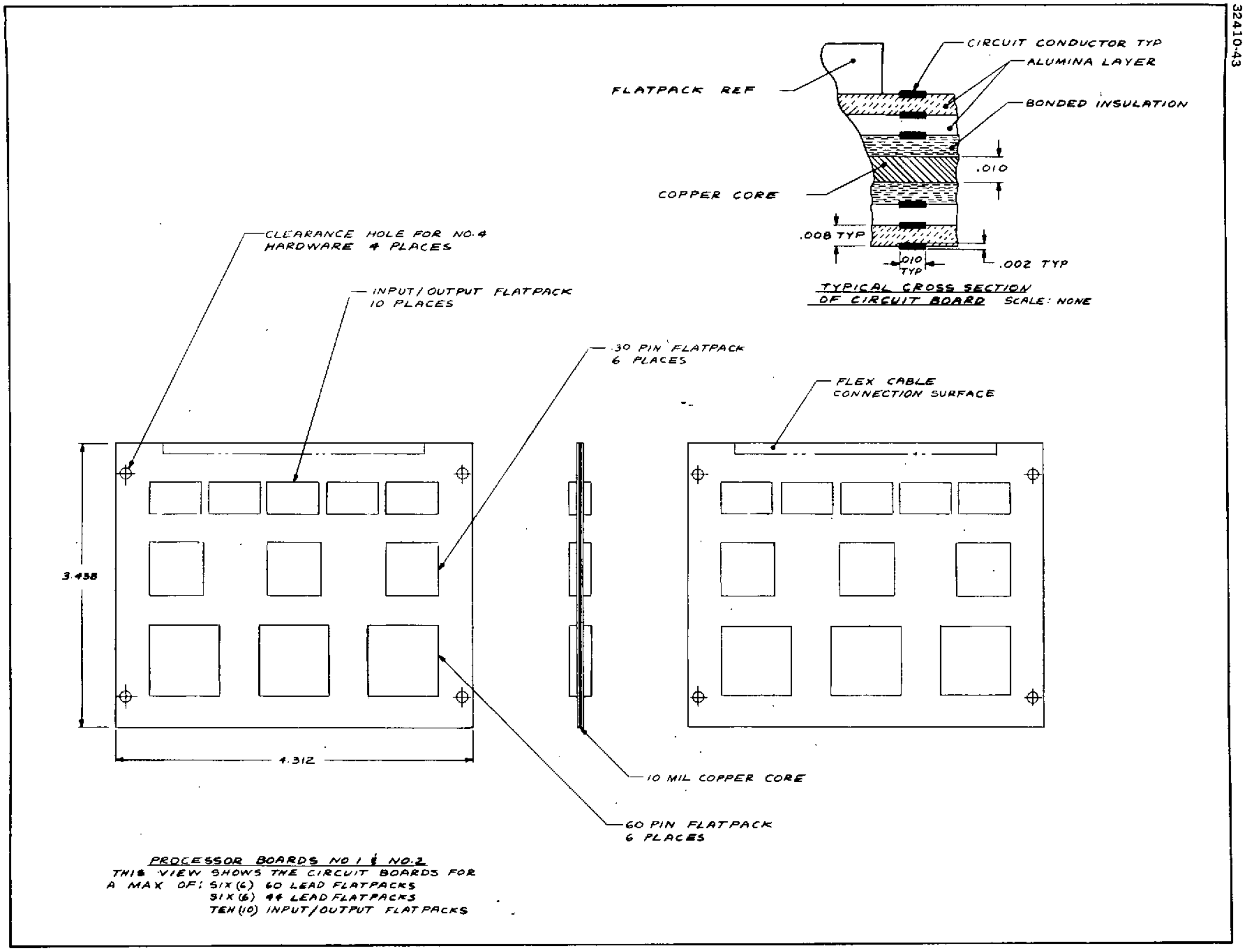

32410-43
CIRCUIT CONDUCTOR TYP
ALUMINA LAYER
FLATPACK REF
BONDED INSULATION
.010
COPPER CORE
.008 TYP
.010 TYP
.002 TYP
TYPICAL CROSS SECTION OF CIRCUIT BOARD SCALE: NONE
CLEARANCE HOLE FOR NO. 4 HARDWARE 4 PLACES
INPUT/OUTPUT FLATPACK 10 PLACES
30 PIN FLATPACK 6 PLACES
FLEX CABLE CONNECTION SURFACE
3.458
4.312
10 MIL COPPER CORE
60 PIN FLATPACK 6 PLACES
PROCESSOR BOARDS NO 1 & NO 2
THIS VIEW SHOWS THE CIRCUIT BOARDS FOR
A MAX OF: SIX (6) 60 LEAD FLATPACKS
SIX (6) 44 LEAD FLATPACKS
TEN (10) INPUT/OUTPUT FLATPACKS

Figure 2. Layout-Processor Circuit Board, ARMMS Computer

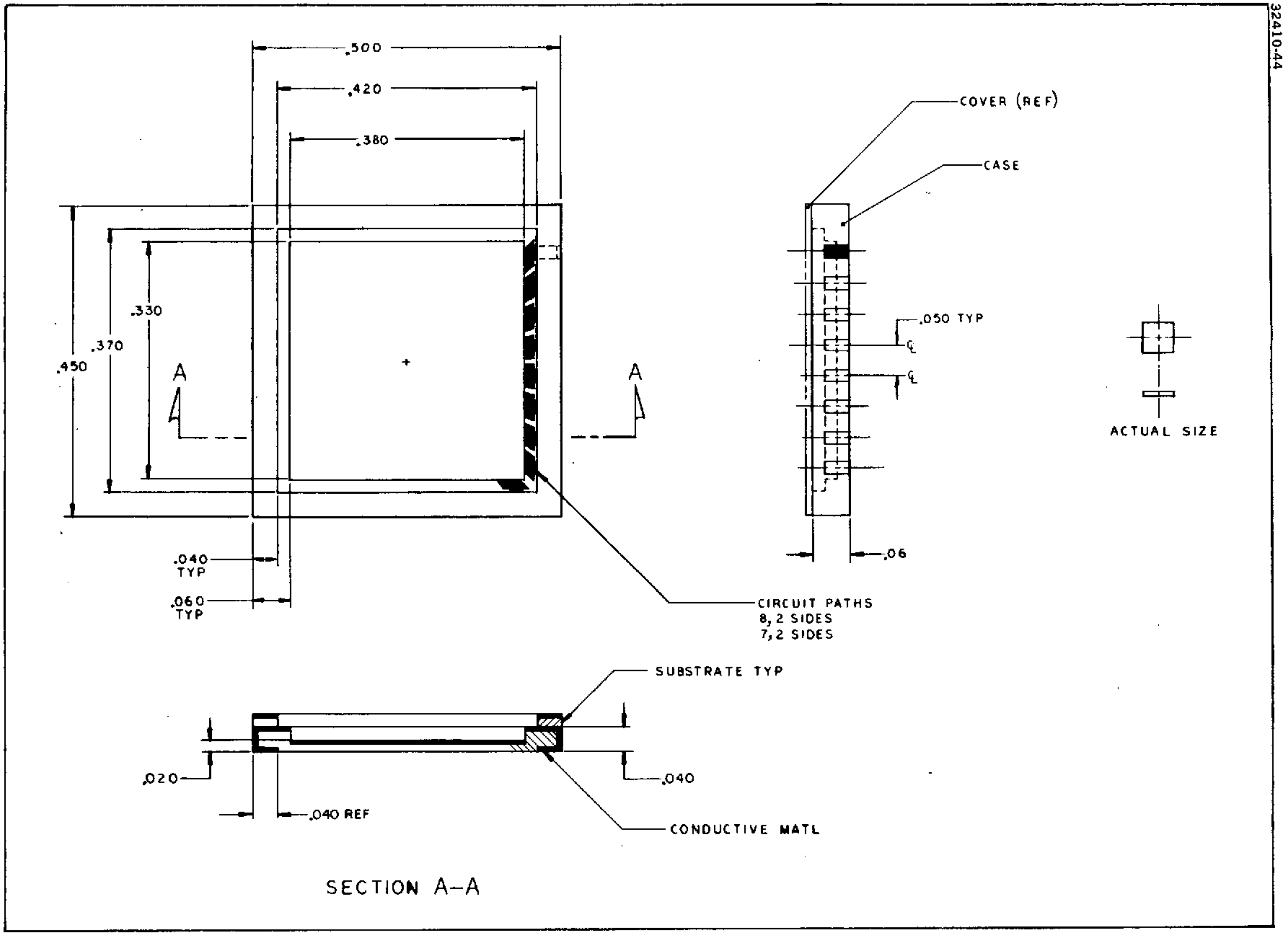


Figure 3. Layout 30 Connection Flatpack Case

The first layer of the circuit card is used for power and ground runs and component attachment. The component contacts total

$$\begin{aligned} 30 \times 3 &= 90 \\ 60 \times 3 &= \underline{180} \\ & 270. \end{aligned}$$

Assuming half of the contact total between components, 45 input conductors and 126 board-to-board conductors (180 pins at 70% efficiency) a total of 283 wiring runs will be required. The total conductors available are 136 + 177 = 313. From this calculation, it can be seen that the cards require 3 layers for each side.

I/O requirements	45
Component-to-component	135
Board-to-board	126
Total conductor requirements	306

The circuit card material will have dielectric constant (K) = 8.6. Assuming six layers of conductors plus a 0.010 inch thick center copper heat sink plane the printed circuit card has a total thickness of 0.06 inch. The spacing between layers will be 0.008 inch, and the conductor width will be as noted above. Therefore the capacitance between the parallel conductors on adjacent layers is 2.41 pf/in. capacitance between layers 1 and 3 parallel conductors is calculated to be 1.47 pf/in. and capacitance between different conductive layers and the heatsink plane ranges from 1.56 to 4.69 pf/in. The module printed circuit card connectors have pin spacing of 0.075 x 0.125 inch, conductor length of 0.650 inch, and pin size of 0.025 x 0.025 inch. The connector body material will be diallyl phthalate with a dielectric constant (K) = 4.5. The capacitance between adjacent pins at 0.075 spacing is 0.33 pf and in the direction of the 0.125 spacing is C = 0.17 Pf.

When a pin is surrounded by ground pins, the worst case condition can occur, which is

$$C = 0.33 + 0.17 + 0.33 = 0.83 \text{ Pf.}$$

5.2.3 Chassis Design

The module chassis will be a machined structure which will provide mechanical support and a thermal path between the printed circuit boards and the unit chassis. The keynote of the module chassis design is its simplicity and the ability to fabricate the part using standard numerically controlled machining equipment. The design also allows for nearly complete assembly and test of the module electronics outside the chassis.

The unit chassis, to which each module is mounted, is also a machined structure. Mating connectors for the modules are attached to this structure as are the system input/output connectors. High power dissipating components (series switch transistors) associated with the preregulator are mounted on machined bases and flanges.

The unit chassis design consists of a basic expandable rectangular configuration. These features provide accommodations for six configurations of module arrangements. It is 15 inches wide, not including the mounting tabs provided as a means of attachment to the spacecraft structure. The length will vary according to a given modular configuration and since only the length is variable economical programming of numerically controlled fabrication equipment is facilitated. The height will be 2.50 inches maximum. The unit chassis and cover are fabricated from 6061-T6 aluminum alloy. The unit chassis contains an intermittent center web which provides a good thermal path from the module attach points to the cold plate of the spacecraft while still allowing interconnections through the mother board. The use of thick side walls and intermittent center webs of the unit chassis was selected for favorable thermal properties. The unit chassis has four circular connectors mounted at one end for input/output signals. For accessibility, a removable cover is provided at the bottom of the unit chassis. An exploded view of a complete unit is shown in Figure 4.

The unit chassis contains a mother board which is configurable to the module arrangements. The mother board is a multilayer printed circuit board containing the interconnecting circuitry to the modules. Data lines are shielded with ground planes above and below and, where necessary, ground shields may be provided between the data lines to eliminate crosstalk.

To interface with the modules, rectangular connectors having 244 sockets are mounted to the mother board. The mother board assembly (with chassis connectors installed) is assembled into the chassis using fillister head screws and locking washers. The modules then mount on the top surface mating the module connector with the chassis connector. A thermal interface material (D-C 340 or equivalent) is used at the structural interface to aid thermal conduction. The thermal requirements have dictated heavy side walls for the modules (0.15") which also satisfies the structural requirements for this resultant cantilever condition. The mother board material will be epoxy glass laminate with a dielectric constant (K) = 5.2. The total board thickness will be approximately 0.060 inch, conductor thickness 0.003 inch, and the conductor width 0.02 inch. The capacitance between conductor and ground planes is 15.0 pf/in. Preliminary investigation indicated a need for approximately 9 layers for the data bus lines and a maximum of three layers for power distribution and control lines. Consequently, the mother board will have 9 to 12 layers.

Intraconnections within the modules will be achieved by an integrated use of multilayered flex cable and multilayered printed circuit boards. Conventional insulated wire will be used only in the power supplies and preregulators. The use of multilayered flex cable provides the advantage of an easy fold-out method for probing or repair of printed circuit cards in their respective modules. Reassembly of the printed circuit cards is a relatively simple task.

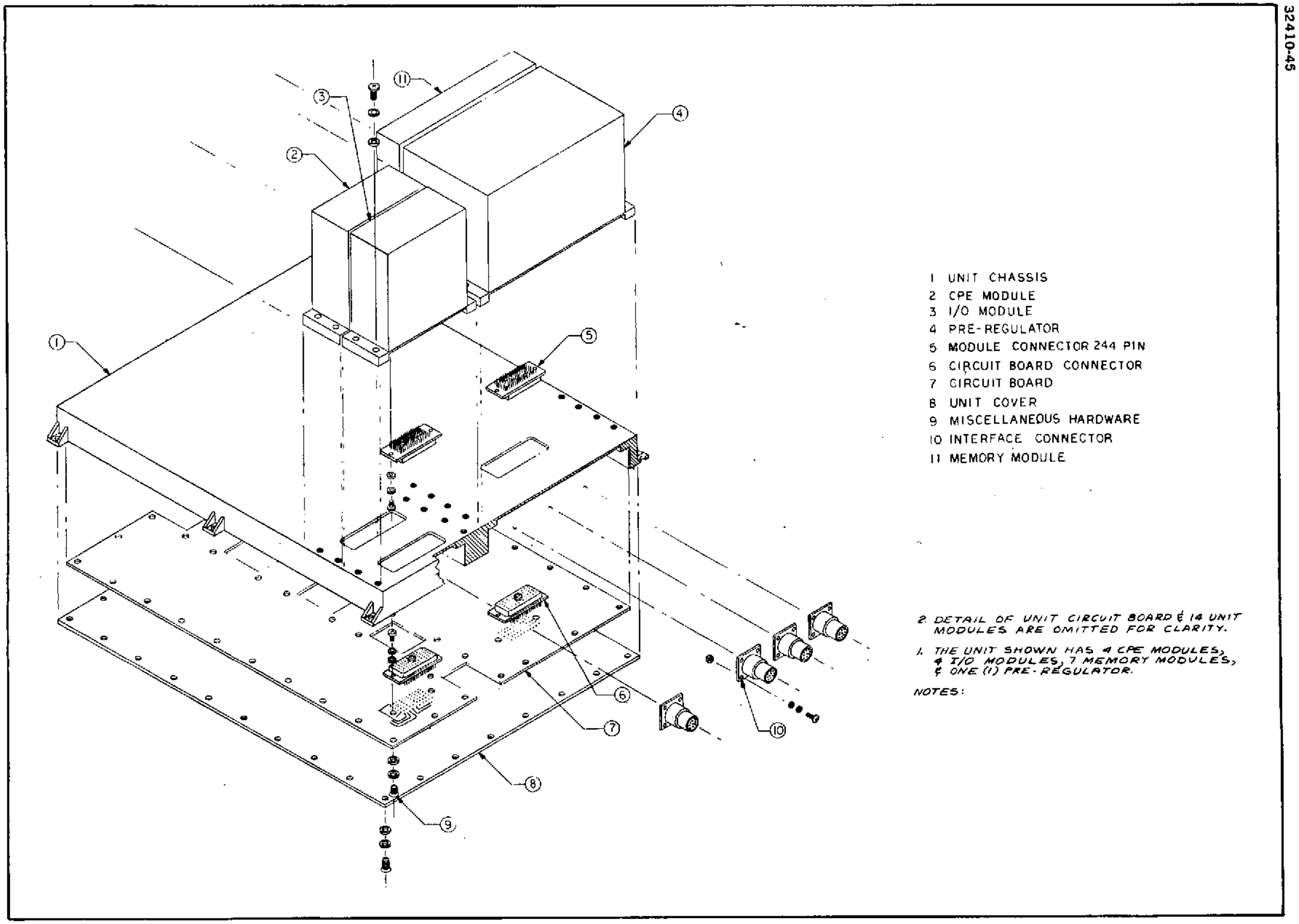

32410-45
1 UNIT CHASSIS
2 CPE MODULE
3 I/O MODULE
4 PRE-REGULATOR
5 MODULE CONNECTOR 244 PIN
6 CIRCUIT BOARD CONNECTOR
7 CIRCUIT BOARD
8 UNIT COVER
9 MISCELLANEOUS HARDWARE
10 INTERFACE CONNECTOR
11 MEMORY MODULE
2. DETAIL OF UNIT CIRCUIT BOARD & 14 UNIT MODULES ARE OMITTED FOR CLARITY.
1. THE UNIT SHOWN HAS 4 CPE MODULES, 4 I/O MODULES, 7 MEMORY MODULES, & ONE (1) PRE-REGULATOR.
NOTES:

Figure 4. Exploded View of ARMMS Unit

A multilayer circuit board will be used to connect signals between boards of a given module. This is accomplished by the use of rectangular connectors mounted on an intraconnection circuit board and subsequently plugged into mating connectors mounted along the edge of each module printed circuit card. This method eliminates the use of wiring between boards and provides a means for simple disconnections. As the design is envisioned, approximately 180 pins will be available on each printed circuit board.

5.2.4 Basic Module Construction

Each module consists of a chassis cover, connector plate and brackets fabricated from aluminum alloy (Figure 5). The modules have a rectangular configuration with external mounting feet. All modules have a height of 4.78 inches. The CPE, and IOP modules have a standard length of 5.88 inches while the memory, BOSS and preregulator modules have a length of 9.00 inches.

5.2.4.1 The IOP and CPE Processor Modules

The IOP and CPE modules are 5.88 inches long by 4.78 inches high by 2.50 inches wide. The circuitry includes 20 ICs in 30-pin packages, 20 ICs in 60-pin packages and 28 input/output flat packs (each containing 8 buffer circuits) packaged on four circuit cards. Also, included is one circuit card containing a DC/DC converter power supply for the module. Power dissipation from the processor cards is 25 watts and from the DC/DC converter is 5 watts. Flexible printed cable provides the output connections from the circuit cards to the 244 pin module output connector. Power distribution within the module is also through the flexible printed cable. The input/output pin requirement for each module is estimated at 225 pins. The current requirement for the power supply implies multiple pins are required for bus power input.

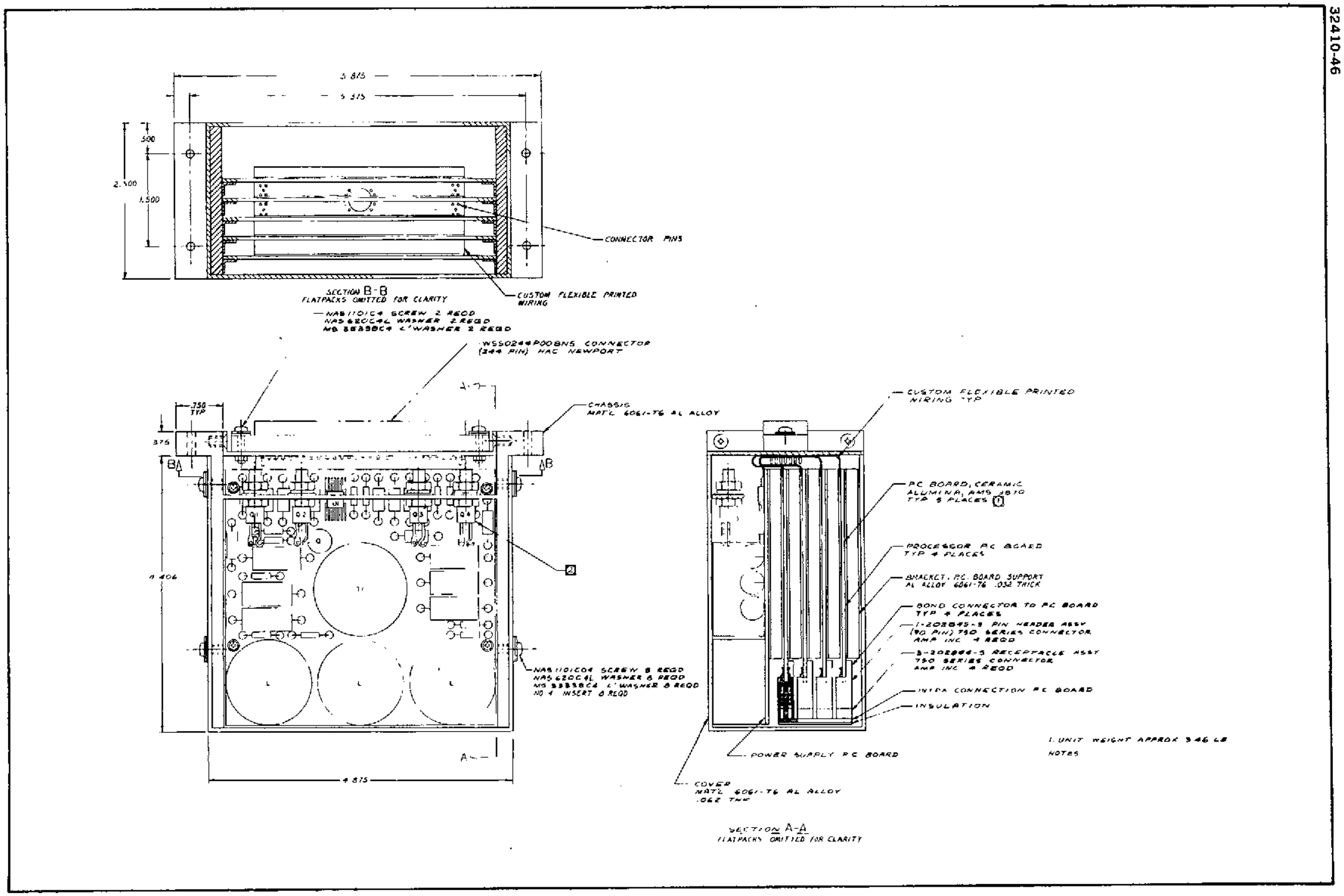
The CPE module is physically identical to the IOP module with the exception of hookup, the elimination of nine input/output flat packs, and 75 input/output pins. An isometric view of a CPE module with power supply is shown in Figure 6. The weight of the IOP or CPE is calculated at 3.46 pounds.

5.2.4.2 The BOSS Module

The BOSS module is packaged very similarly to the CPE module with the exception that it contains 20 circuit cards and two DC/DC converter cards. Dimensionally, it is 9.00 inches long x 4.78 inches high and 4.95 inches wide. There are two 244-pin connectors for connection to the base assembly. The module contains 160 integrated circuits. Eighty of these devices are the 30-lead type, while 80 are the 60-lead type. In addition, there are two crystal oscillator packages. There will be 300 input-output lines interfacing with discrete (hybrid) buffer circuits. A total of 76 I/O buffer flat packs will be required. Total power dissipated within the module will be

60 watts logic power
10 watts bus interface
14 watts DC/DC converter
84 watts total.

The BOSS module is estimated to weigh 10.6 pounds (Figure 7).


5.875
5.375
.500
2.500
1.500
CONNECTOR PINS
SECTION B-B
FLATPACKS OMITTED FOR CLARITY
CUSTOM FLEXIBLE PRINTED WIRING
NAS1101C4 SCREW 2 REQD
NAS620C4L WASHER 2 REQD
MS 35338C4 L'WASHER 2 REQD
WS50244P008NS CONNECTOR (244 PIN) HAC NEWPORT
.750 TYP
.375
CHASSIS MATL 6061-T6 AL ALLOY
B
B
A
A
4.406
4.875
Q1
Q2
Q3
Q4
2
NAS1101C04 SCREW 8 REQD
NAS620C4L WASHER 8 REQD
MS 35338C4 L'WASHER 8 REQD
NO 4 INSERT 8 REQD
CUSTOM FLEXIBLE PRINTED WIRING TYP
PC BOARD, CERAMIC ALUMINA, AMS 3870 TYP 5 PLACES 1
PROCESSOR PC BOARD TYP 4 PLACES
BRACKET, PC BOARD SUPPORT AL ALLOY 6061-T6 .032 THICK
BOND CONNECTOR TO PC BOARD TYP 4 PLACES
1-202845-5 PIN HEADER ASSY (90 PIN) 750 SERIES CONNECTOR AMP INC 4 REQD
3-202844-5 RECEPTACLE ASSY 750 SERIES CONNECTOR AMP INC 4 REQD
INTRA CONNECTION PC BOARD
INSULATION
POWER SUPPLY PC BOARD
COVER MATL 6061-T6 AL ALLOY .062 THK
SECTION A-A
FLATPACKS OMITTED FOR CLARITY
1. UNIT WEIGHT APPROX 3.46 LB
NOTES

32410-46

Figure 5. Layout-Power Supply and Processor Module

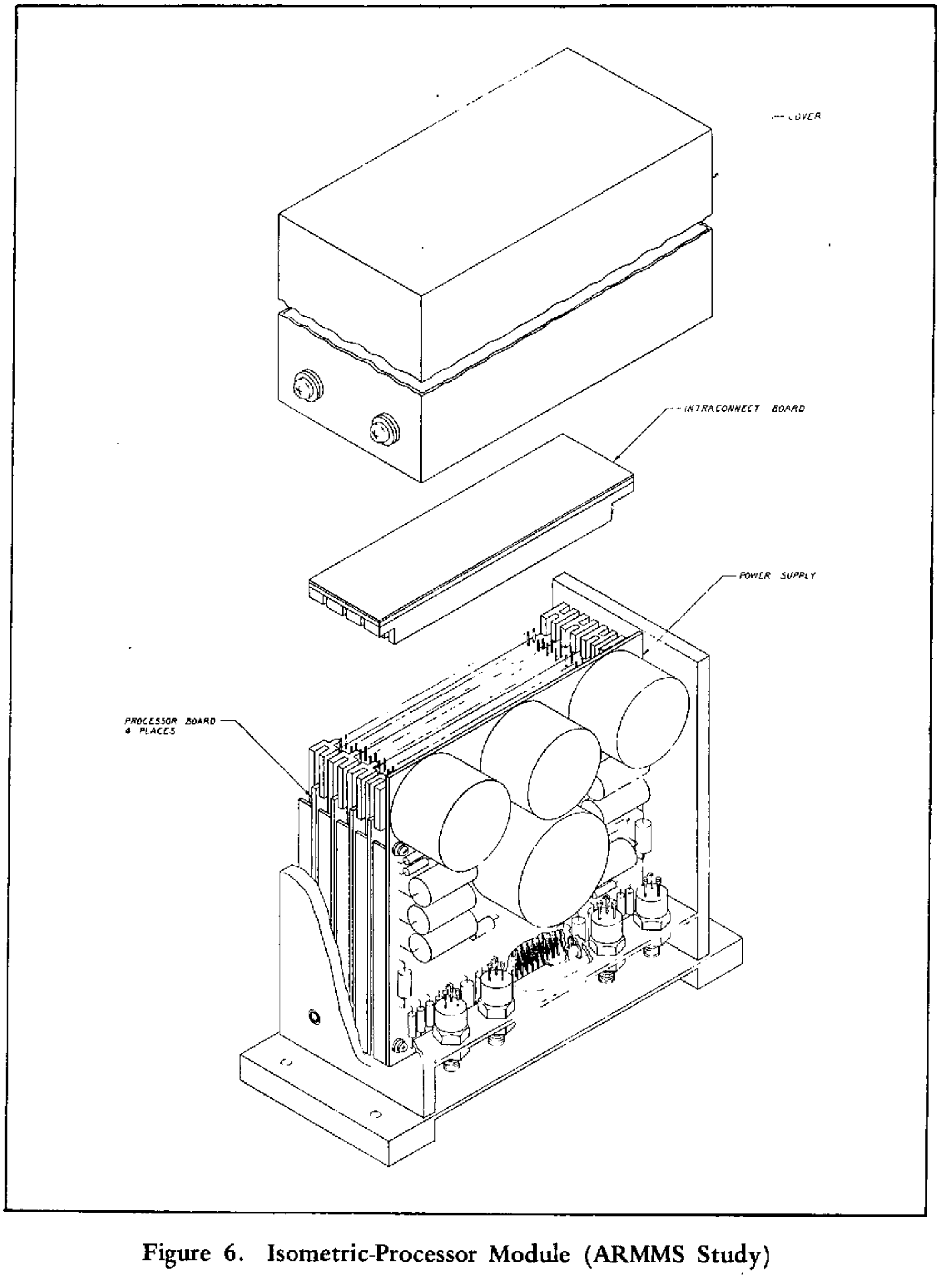
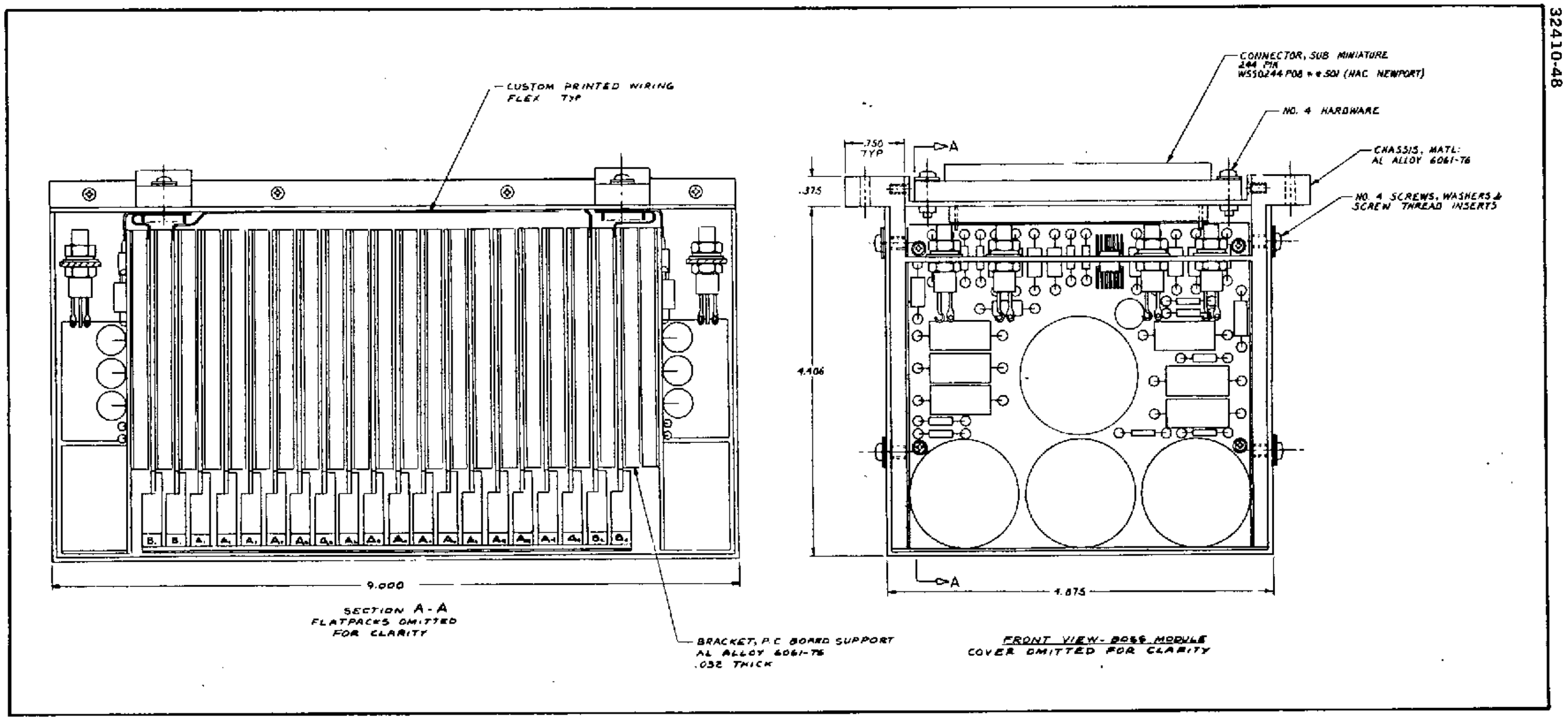


Figure 6. Isometric-Processor Module (ARMMS Study)



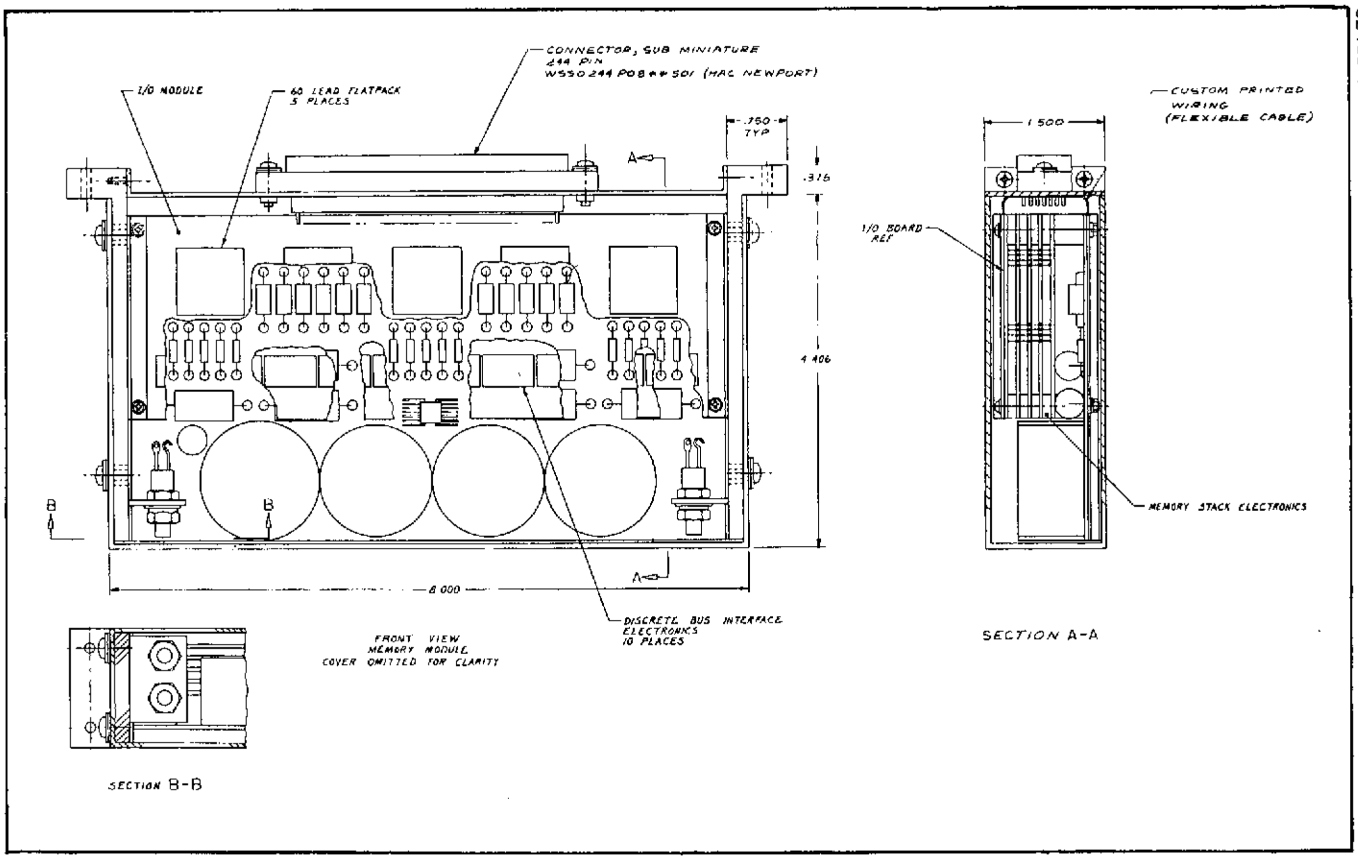
32410-48
CUSTOM PRINTED WIRING FLEX TYP
CONNECTOR, SUB MINIATURE 244 PIN WS50244 P08 ✱✱.501 (HAC NEWPORT)
NO. 4 HARDWARE
.750 TYP
A
CHASSIS, MATL: AL ALLOY 6061-T6
NO. 4 SCREWS, WASHERS & SCREW THREAD INSERTS
.375
4.406
9.000
SECTION A-A FLATPACKS OMITTED FOR CLARITY
BRACKET, P.C BOARD SUPPORT AL ALLOY 6061-T6 .032 THICK
4.875
FRONT VIEW - BOSS MODULE COVER OMITTED FOR CLARITY

Figure 7. Layout-BOSS Module

5.2.4.3 The Memory Module

The memory module consists of a four card plated wire memory stack with an input/output card mounted on either side (Figure 8). The entire stack is bolted together with spacers maintaining proper distance between cards. Like all the other modules, the memory module contains its own DC/DC converter. The module is 9.00 inches long x 4.78 inches high x 1.50 inches wide. Estimated weight is 4.03 pounds. The memory module contains ten integrated circuits, each with 60 leads. There will be 150 interface lines with the same buffer as used on the CPE module. The memory will be approximately 320 K bits organized as 8 K words, 40 bits per word. The total power dissipation within the module shall be as follows:

5	watts logic
15	watts stack electronics
5	watts bus interface
5	watts DC/DC converter
30	watts total

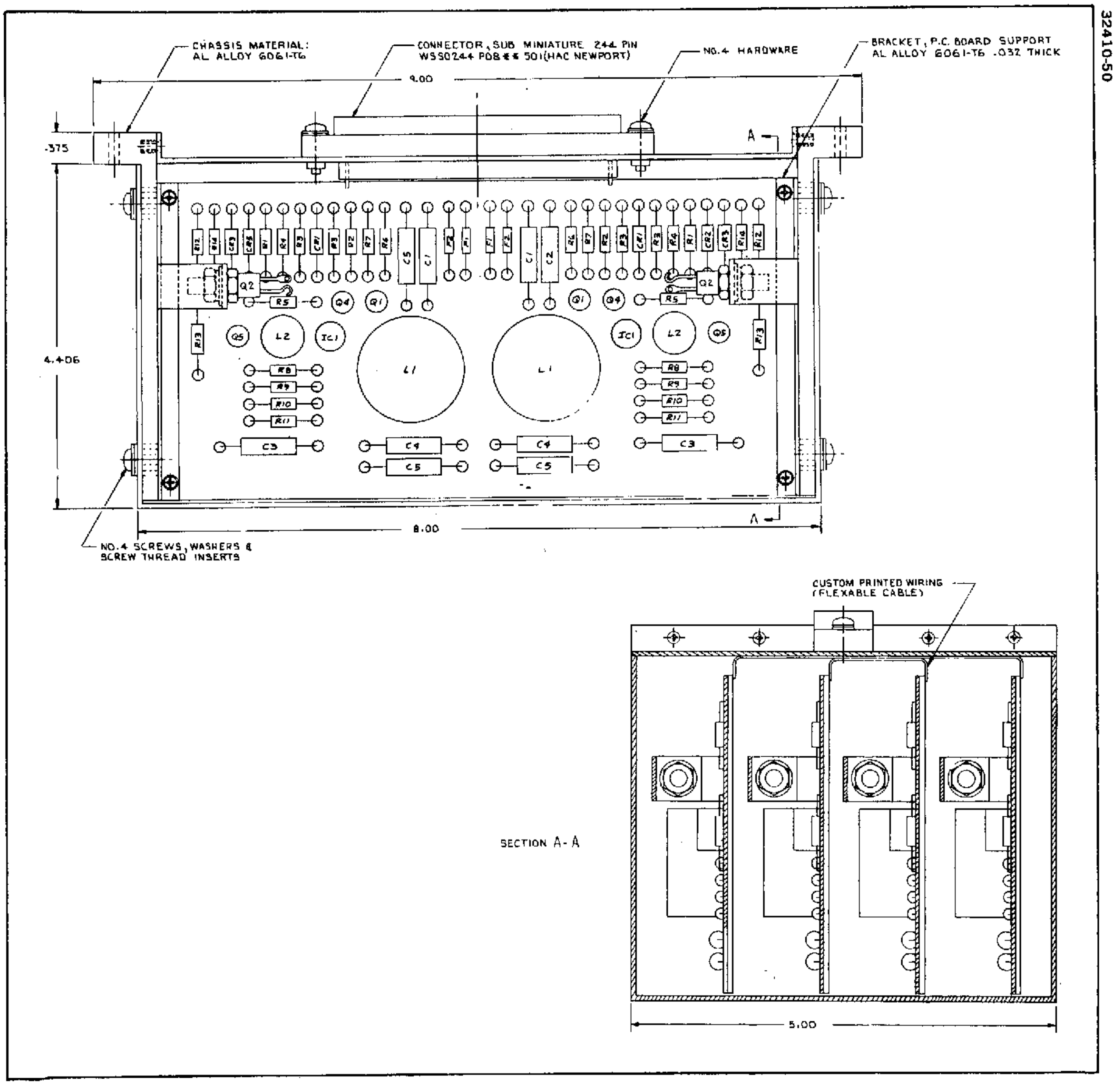


CONNECTOR, SUB MINIATURE 244 PIN WSSO 244 PO8 ** 501 (HAC NEWPORT)
I/O MODULE
60 LEAD FLATPACK 5 PLACES
.750 TYP
.376
A
4.406
B
B
8.000
A
FRONT VIEW MEMORY MODULE COVER OMITTED FOR CLARITY
DISCRETE BUS INTERFACE ELECTRONICS 10 PLACES
SECTION B-B
1.500
CUSTOM PRINTED WIRING (FLEXIBLE CABLE)
I/O BOARD REF
MEMORY STACK ELECTRONICS
SECTION A-A
32410-49

Figure 8. Layout-Memory Module

5.2.4.4 The Preregulator Module

The preregulator module (Figure 9) consists of one to four printed circuit cards. Each card contains the equivalent of two preregulators. The number of printed circuit cards in each module will vary according to the requirements of the system configurations. With a maximum configuration of eight preregulators, the module is 9.00 inches long by 4.78 inches high by 4.95 inches wide. Estimated weight is 7.00 pounds. Total power dissipation per preregulator is 22 watts, of which 8 is dissipated in the line switch which is mounted to the unit chassis.



CHASSIS MATERIAL: AL ALLOY 6061-T6
CONNECTOR, SUB MINIATURE 244 PIN WSS0244 P08** 501(HAC NEWPORT)
NO.4 HARDWARE
BRACKET, P.C. BOARD SUPPORT AL ALLOY 6061-T6 .032 THICK
9.00
.375
4.406
8.00
A
NO.4 SCREWS, WASHERS & SCREW THREAD INSERTS
CUSTOM PRINTED WIRING (FLEXABLE CABLE)
SECTION A-A
5.00
32410-50

Figure 9. Pre-Regulator Module Max Configuration

5.2.5 Weight, Volume, Mass Properties

Tables I and II itemize the weight, volume and size of the five individual ARMMS modules and six possible computer combinations of Table III. Table IV lists module component totals. Three of these configurations are illustrated in Figures 10, 11, and 12.

TABLE I. MODULE MASS PROPERTIES

Module	Size inches (cm)	Volume in.^{3} (cm^3)	Weight pounds (KG)
CPE	2.5 x 4.78 x 5.88 (6.35) x (12.14) x (14.93)	70.3 (1150.93)	3.46 (7.61)
IOP	2.5 x 4.78 x 5.88 (6.35) x (12.14) x (14.93)	70.3 (1150.93)	3.46 (7.61)
Memory	1.5 x 4.78 x 9.00 (3.81) x (12.14) x (22.86)	64.5 (1057.35)	4.03 (8.86)
BOSS	4.95 x 4.78 x 9.00 (12.57) x (12.14) x (22.86)	213.0 (3488.43)	10.6 (23.32)
Preregulator	5.0 x 4.78 x 9.00 (12.70) x (12.14) x (22.86)	215.1 (3524.50)	7.00 (15.4) Max. Conf.

TABLE II. ARMMS COMPUTER MASS PROPERTIES

Configuration	CPE	IOP	BOSS	Memory	Weight Pounds	Volume in.^{3}
1	1	1	-	2	33	945
2	2	2	-	4	51	1280
3	3	3	-	6	70	1740
4	4	4	-	8	90	2320
5	4	4	1	16	140	3400
6	7	4	1	25	194	5600

TABLE III. ARMMS COMPUTER MODULAR CONFIGURATION

Configuration	Preregulator	CPE	IOP	Memory	BOSS	Total Module Count
1	1	1	1	1-2	-	4-5
2	1	2	2	2-4	-	7-9
3	1	3	3	3-6	-	10-13
4	1	4	4	4-8	-	13-17
5	1	4	4	8-16	1	18-26
6	1	7	4	8-25	1	21-38

TABLE IV. ARMMS MODULE PHYSICAL CHARACTERISTICS

Module	No. of 30 Pin IC's	No. of 60 Pin IC's	No. of Hybrids	No. of Interface Lines	No. of PC Boards	Power Dissapation
CPE	20	20	19	150	4	30
IOP	20	20	28	225	4	30
BOSS	80	80	76	300	20	85
Memory	--	10	19	150	6	30

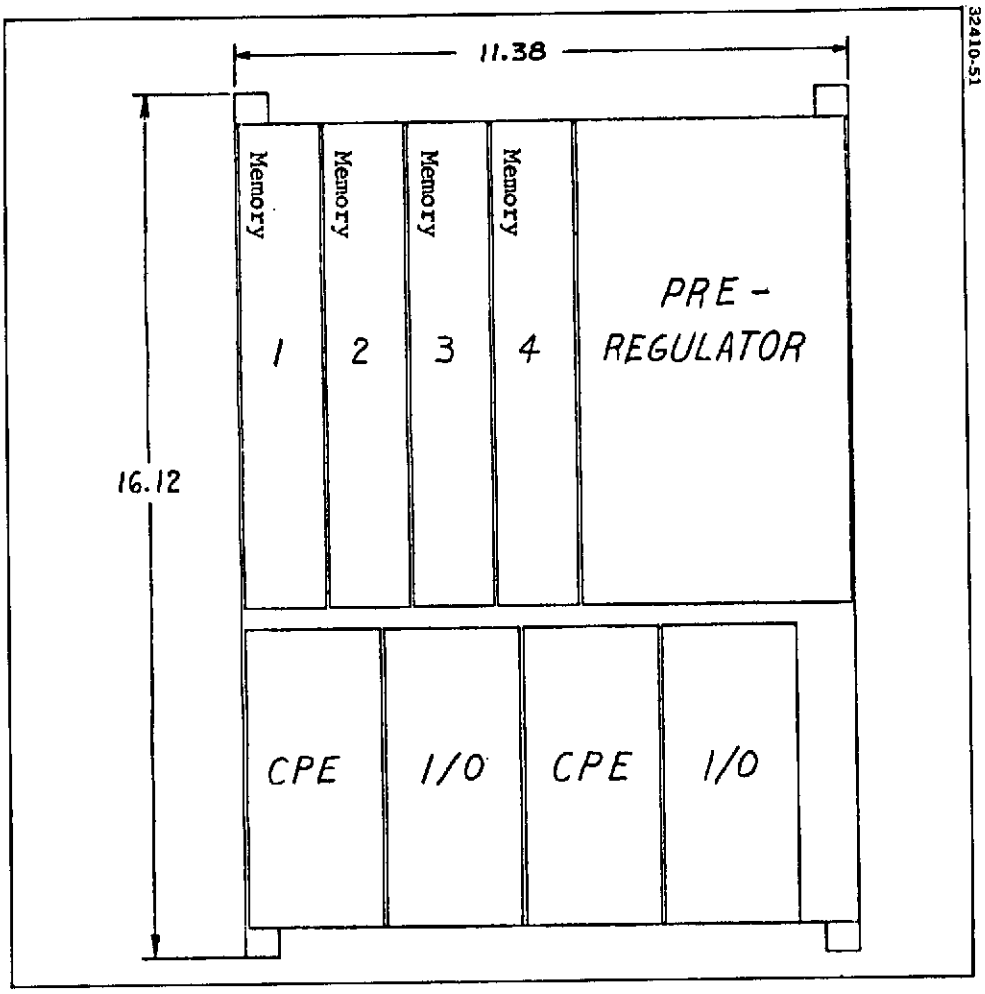


Figure 10. Configuration No. 2 of ARMMS Chassis

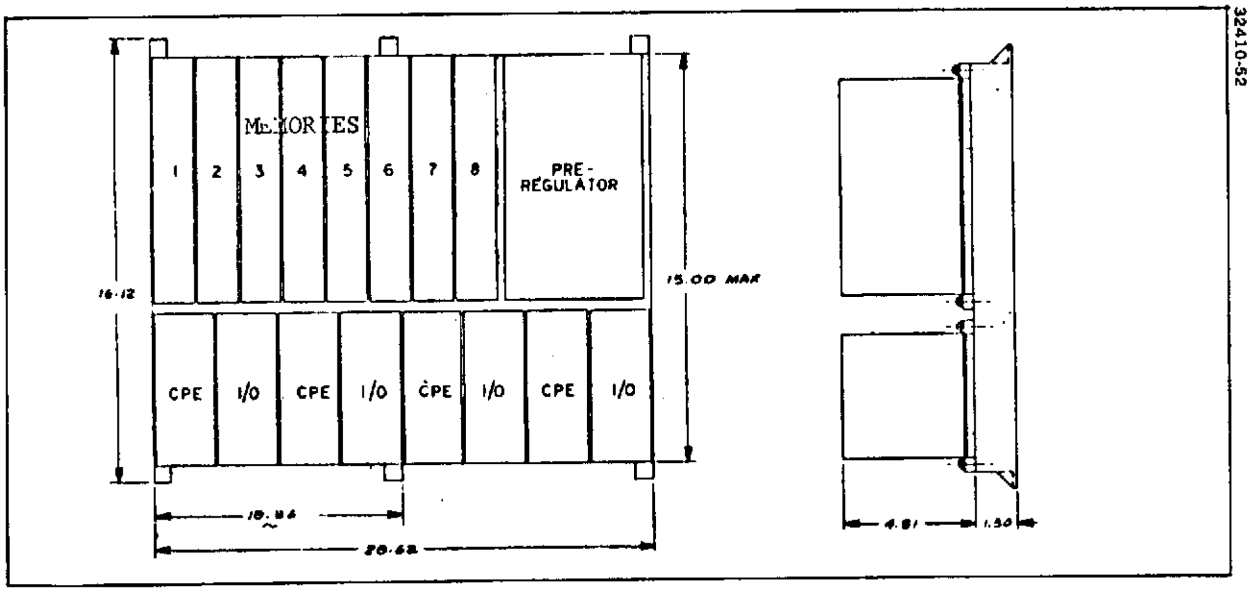


Figure 11. Configuration No. 4 of ARMMS Chassis

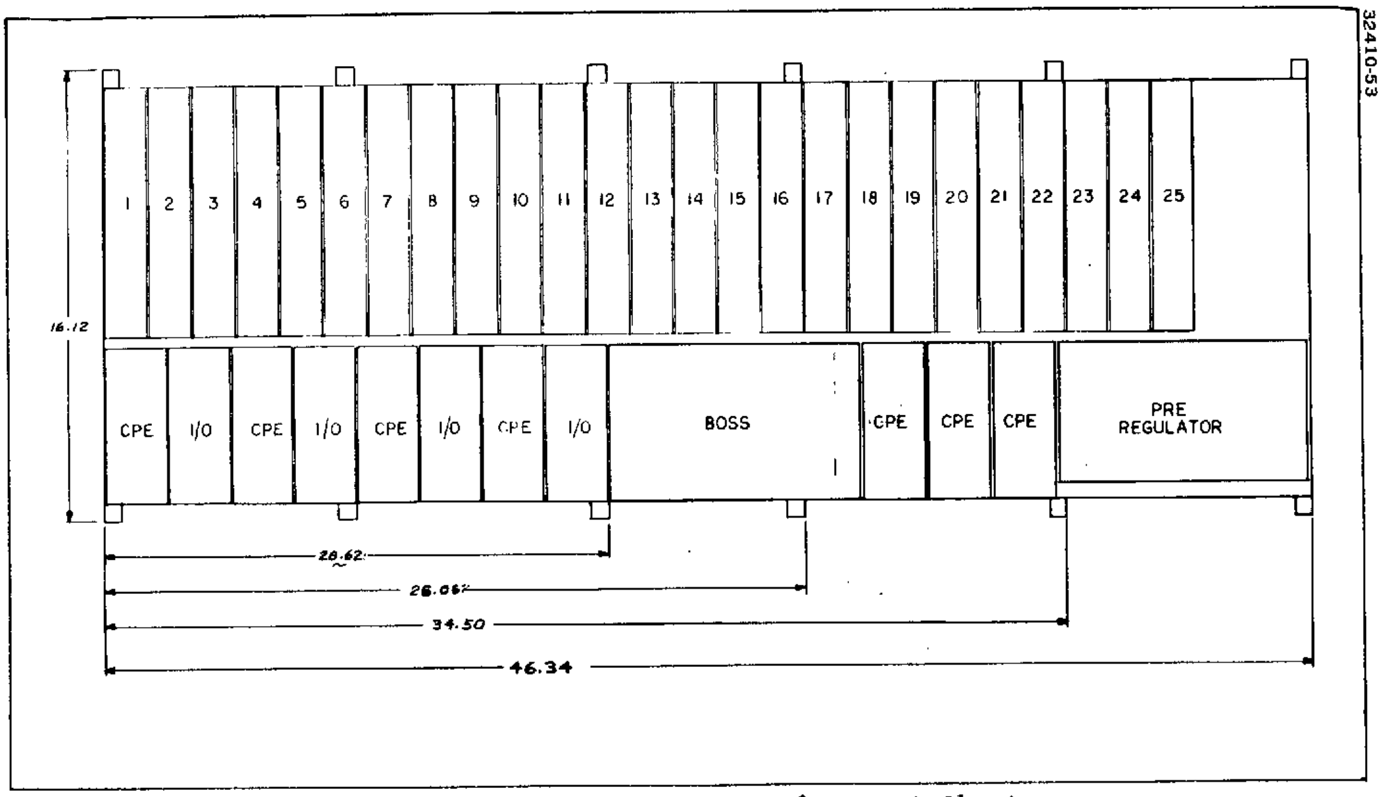


Figure 12. Configuration No. 6 of ARMMS Chassis

5.2.6 Thermal Analysis

Analysis of the proposed design indicates that the greatest temperature rise between the baseplate and the hottest component is 88°C. This will allow the unit to be mounted on a 37°C baseplate while maintaining a maximum chip temperature of 125°C. Significant reductions in the thermal rises of the unit may be achieved by improving the thermal conductivity of interfaces among the structural members of the unit. Detailed investigation of contact interface phenomena will be necessary before a final hardware implementation can be achieved.

The ARMMS unit is intended to be operated in a space environment which eliminates convection as a potential mode of heat transfer. It is assumed that the chassis is operated in an environment such that adjacent modules and printed boards are at the same temperature level, and radiation heat transfer is negligible. Except for the utilization of heat pipes, conduction heat transfer is the only heat removal mode considered in this study.

Heat generated by the chips will be transfered by conduction through the case of the flat packs to the printed circuit boards. The ceramic alumina printed circuit board is assumed to have a thermal conductivity (k) of 14.5 (BTU/HR-ft^2 - °F/ft) which is a typical value for this material. Some kind of heat sinking device (copper plate k = 200 BTU/HR-ft^2 - °F/FT or heat pipes) will be required to enhance the conduction path to the edges of the printed circuit board. The heat is conducted from the printed circuit board through mounting brackets to the aluminum sides of the modules, across several contact resistances in the aluminum chassis, and finally rejected to the constant temperature cold plate. The results of this study are in the form of temperature rising steps from the cold plate and are presented in Table V.

TABLE V. EXAMPLE FOR ESTIMATING TOTAL CHIP TEMPERATURE RISE (ALL TEMPERATURES °C)

Parameter		I/O, CPE and Memory Section			Boss and Memory Section	
		I/O	CPE	Memory	BOSS	Memory
Cold plate (reference temp.)		0	0	0	0	0
Module base temp. rise	Resistivity = 0.001 (Hr-ft^2-°F/BTU)	28	28	32	30	28
Module wall temp. rise	Wall thk. = 0.15"	12	29	22	29	22
PC board temp. rise across bracket	Resistance = 0.4 (Hr-°F/BTU)	4	3	5	3	5
PC board temp. rise	Copper plate thk. = 0.01"	12	8	28	8	28
Chip temp. rise across flat pack		5	5	5	5	5
Total chip temp. rise °C above cold plate		61	73	63	75	88

5.2.7 Stress Analysis

The proposed ARMMS module design has not been subjected to a detailed stress analysis because the vibrational environment for the proposed system is undefined. However, a unit similar in design but slightly smaller in physical size than the CPE module has been fabricated and vibrationally tested with no mechanical or electrical anomalies. The unit was subjected to a sine sweep vibration test of 10 G's from 20 Hz to 2 kHz for nine minutes per axis in each of three axes, plus a random vibration test of 0.2 g^2/Hz between 20 Hz and 2 kHz for four minutes per axis in each of three axes. The RMS level of the random vibrational test is approximately 20 G's. The unit was electrically powered and monitored during each test. This environment represents a typical G load which a unit mounted within a spacecraft could be expected to see during launch.

Presently under development is a small (approximately CPE module size) high density unit design to withstand a random vibration level of approximately 30 G's RMS. If significantly higher power spectral densities are expected for ARMMS, additional testing and analytical stress calculations will be necessary to ensure that unit's ability to withstand the higher vibrational loads. If a particular launch vehicle power spectral density curve is available, it may be possible to design the modules such that all major resonant points of the structural assemblies lie outside the frequency spectrum of acoustical or mechanical vibration energy of that particular vehicle.

5.2.8 Areas That Could Use Additional Investigation

One of the major problems in the conceptual design just presented is the buildup on connector extraction forces in the card intraconnection area. There are zero insertion/extraction force connectors on the market, but they are not compatible with this design. Possibly in the 1980 era, there will be usable connectors for this application. If this does not prove to be the case, a special insertion/extraction mechanism can be designed for the connectors proposed.

The BOSS module, as presently envisioned, is partitioned to provide the greatest utilization of the CPE printed circuit board assemblies. By eliminating the standardization of the printed circuit board design between the CPE and BOSS, one "A" partition board can be eliminated per "A" partition. This would result in the elimination of four printed circuit board assemblies in the BOSS and a one inch reduction in its length.

SECTION 6

ARMMS RELIABILITY STUDIES

This section consists of two parts. The first summarizes the reliability data base study performed during Phase I which yielded the failure rate numbers used in the module reliability analyses performed during Phase III and described in Section 4 of this report. Equations for hand calculating ARMMS reliability using the numbers from Section 4 are also given. The second topic surveys reliability studies performed elsewhere, assessing their degree of applicability to ARMMS, and then describes a new model developed specifically for ARMMS. This model will require programming on a digital computer before it can be used but is of considerable theoretical interest in its present form since it points up some of the unique modeling problems presented by the ARMMS architecture.

SECTION 6

ARMMS RELIABILITY STUDIES

6.1 RELIABILITY DATA BASE

This topic summarizes the component reliability data used elsewhere in this report for analyses and predictions. The rates are for high quality electronic parts screened for space environment at low levels of electrical, thermal, and mechanical stress. The rates which appear in Table I are projections for 1973 technology based on 1970 handbook predictions viewed in the light of Hughes experience with space programs. A more complete table appeared in the Phase I report. It would have been desirable to accumulate and analyze part failure data from many space programs and use the resulting best estimates for the ARMMS data base. This proved to be infeasible because while millions of system hours of data from space programs exist the data is still too scanty and uncertain to accurately access the failure rate of an individual part type such as a power wire wound resistor because the accuracy of failure rate estimates, especially in the zero-failure case, depends heavily on the amount of time accumulated. The problem is compounded by the fact that isolation of failures to a piece part through limited telemetry data is often impossible so many part failures never get charged against the part. Finally much of the data is several years old and reflects now outdated technology.

Therefore, a different technique was used, whereby part failure rates were estimated using a combination of observed and predicted values. This was based on the two ideas: 1) that a handbook predicted value (which is partly theoretical) is better than an observed value if the data for the observed value is scanty; and 2) an estimate based on a prediction and an observation is better than one based only on observation. Hughes Aircraft Company has logged many

TABLE I. SELECTED ARMMS FAILURE RATES

Part Type	Failure Rate/10^6 Hr
Integrated Circuits (250 gates CMOS)	0.025
Integrated Circuits (Linear)	0.009
Capacitor (Glass)	0.00016
Capacitor (Film)	0.00008
Capacitor (Solid Tantalum)	0.00008
Diode (Power Rectifier)	0.0008
Diode (Switching)	0.0008
Diode (Zener)	0.0016
Resistor (Carbon Composition)	0.00002
Resistor (Thick Film)	0.0006
Resistor (Power Wire Wound)	0.0042
Transistor (Power)	0.0044
Transistor (High Speed Switch)	0.0016
Plated Wire	0.00008

hours in space with its various satellite programs, yielding useful data for reliability predictions of space systems. The expected number of failures was predicted for all satellites to date, and actual failures were then monitored, although not always classified. Thus, it was possible to compare predicted and observed failure rates at the system level, and determine the proper modifying factor for predicted failure rates. While the observed number of failures so far has been 47% of the predicted number, the predicted numbers are used in the table and throughout this report to provide an extra margin of safety. These numbers were multiplied by 0.47 in the table in the Phase I report however.

Little is known about the failure rates of parts in unenergized systems. The best available ratio, from a 1971 report by Aerospace Corp. for "hi-rel parts with rigorous specifications, stringent manufacturing controls, and extensive screening" is 0.8. This number is higher (i.e., more pessimistic) than for MIL-STD or commercial parts which involve lesser manufacturing controls and whose operating to dormant failure rate ratios tend to appear more often in the literature. It is also generally assumed that high rates of power cycling are detrimental but that very low rates (1 cycle/1000 hours) have a neglible effect. Thus cycling is assumed not to effect failure rates in ARMMS. Failure rates are assumed to remain constant throughout the mission allowing the use of the usual exponential probability function. Finally to take into account the rapid reduction in MOS-device failure rates the microsecond failure rates have been multiplied by a factor of 0.25 which seems conservative for extrapolating these rates between 1970 and 1973. No improvement has been assumed in other failure rates over this period.

Once a composite failure rate has been obtained for a module, based upon the above discussion and the module's design and taking into account fault detection and masking if any, it is possible to compute the probability of successful operation of the module or of a given number out of a group of modules either using the computer model described in the next topic or, if active and passive failure rates are taken to be the same, by the equation below:

If $q = e^{-\lambda t}$ = probability of a module with failure rate λ is still operating at time t and $p = 1-q$

Then $Q_{mn} = \left\{ n!/(n-m)!m! \right\} q^m p^{(n-m)}$ = probability that m out of n modules each with failure rate λ will operate at time t

Therefore $P_f = 1 - \sum_{k=0}^{m} Q_{mn}$ = probability that no more than n-m-1 modules have failed by time t

6.2 THE RELIABILITY MODELING OF COVERAGE IN THE SIMPLEX, MULTIPROCESSOR, DUPLEX AND TMR MODES FOR FAULT TOLERANT COMPUTER SYSTEMS

I. Introduction

The purpose of the present analysis is to develop a model of fault-tolerant computer coverage for the Simplex, Multiprocessor, Duplex and TMR modes. This model will allow for differing active and passive hazard rates λ, μ where $\lambda \geq \mu$ and fault detection and fault masking capability in which each module of a specified module class can tolerate up to one maskable failure.

The present study will cover a mission of one phase duration as opposed to the more general type of model, developed in [1], in which a multi-phase mission was analyzed. In that study it was assumed that coverage was perfect, i.e. the fault detection, isolation and restoration of failed modules could be achieved with probability equal to one. Then it was possible to examine an entire mission profile of computer phased activities, in which each phase was describable in terms of two integers N_i and D_i, where N_i was the desired number of modules which the mission planner required for mission phase i, while D_i was the minimal number of modules of the given module class that must remain operational, in order to insure that the module class will perform its essential functions for that given phase. If the actual number of non-failed modules was N_i^*, where $D \leq N_i^* \leq N$, at the beginning of phase i, then, of course, the module class was run with N_i^* instead of N_i units (a unit will be used as a synonym for a module in this analysis). When the single phase analysis is completed it will be appropriate to examine the more difficult question of phase composition in the presence of undetectable and maskable type faults.

The modeling effort is accomplished by using a state space approach and employing the method of birth-death processes. No simpler approach appears feasible and, in fact, an elementary argument developed by Bouricius, etal, [2], for the special case $\lambda = \mu$ and (in our notation) $P_m = P_d$, i.e. the probability of fault detection = probability of fault masking, turns out to be in error for the case of multiple fault tolerance (i.e. for $f \geq 1$ in their notation for ${}_c^fR_S^1$, which represents the module class reliability of a simplex operated system).

The major conclusion that emerges from the present analysis is that the conservative statements of Bouricius, etal. appearing in [2], [3], [4] may be significantly improved upon in the ARMMS context since in their development of coverage, f, the number of faults per module, was always assumed to be equal to zero. The present treatment which performs a deeper analysis of the coverage mechanism allows $f = 1$ and thus, for the same parameter values which Bouricius uses, one should expect a measurable improvement in reliability. The present method allows for quantitative comparison of the Bouricius model v.s. the one presented here.

As an example, consider a multiprocessor system consisting of N active units with no spares and assume that the failure rate due to maskable type faults is λ_m so that the probability that a given fault is maskable is given by $Pm = \lambda_m/\lambda$. Then, in the Bouricius analysis, since no faults are allowable, one would obtain for reliability of this system for the time T:

$$Rel_1 = e^{-N\lambda T}$$

However, through the introduction of a double error detecting, single error correcting Hamming code we would obtain for the system reliability:

$$Rel_2 = (1 + \lambda_m T)^N e^{-N\lambda T}$$

For small values of T the two results would be approximately equal with $Rel_1 < Rel_2$, always. For moderate values of T it is clear, though, that significant reliability improvement can be achieved for Rel_2 relative to Rel_1, i.e.

$$\frac{Rel_2}{Rel_1} = (1 + \lambda_m T)^N > 1 + N \lambda_m T$$

As spares are introduced the situation becomes more complex and, as will be presented in the development, it is essential to relate the notion of coverage to the actual design or architectural implementation of the fault tolerant mechanisms employed by the computer system.

In Section II, below, a review of pertinent work in the coverage area will be presented and in Section III the underlying assumptions and mathematical development of the present coverage model will be given. Actual numerical evaluations of the present model are planned once the system of birth-death differential equations describing reliability performance have been programmed.

II. Review of Previous Studies in Coverage Analysis

A. The Case of Perfect Coverage, $c = 1$

Kletsky, [9], in 1962, was an early contributor to the problem of system reliability evaluation in the presence of differing power-off and power-on hazard rates μ, λ, respectively. He treated the case of an active-passive standby system in which N modules are always active and which possesses S standby spares. This system of $M = N + S$ identical modules will be considered to have failed if the number of available good units falls below N. Introducing, for convenience, the notation (due to Bouricius, et. al., [2]) ${}^{f}_{c}R^{N}_{S}(\lambda, \mu; T)$ = Prob (System is good through the time interval $[0, T]$ when the number of tolerable faults/module is equal to f, the coverage is equal to c and the hazard rates are λ, μ), Kletsky obtained, using Laplace transform methods:

$$ {}^{0}_{1}R^{N}_{S}(\lambda, \mu; T) = e^{-N\lambda T} \sum_{k=0}^{S} \binom{k - 1 + NK}{k} \left(1 - e^{-\mu T}\right)^{k} \qquad \text{Eq. 1} $$

Here $K = \lambda/\mu$ and is an important design parameter of fault tolerant computer systems, [11].

In [5], Mathur made a similar study for the reliability of a TMR system with S spares to obtain, assuming perfect coverage:

$$ \begin{aligned} R(3, S)(T) = R_{TMR/S} &= \text{Rel(TMR system with S spares)} \\ &= 3R^{2} \left\{ \prod_{i=0}^{S-1} \frac{3K + S - i}{(K + S - i)} - \frac{2RK^{2}}{S!} \left[\prod_{i=0}^{S-1} (3K + S - i) \right] . \qquad \text{Eq. 2} \right. \\ &\quad \left. \cdot \left[\sum_{i=0}^{S} \binom{S}{i} \frac{(-R_{S})^{S-i}}{(K + S - i)(3K + S - i)} \right] \right\} \end{aligned} $$

where

$$ R = e^{-\lambda T} \text{ and } R_{s} = e^{-\mu T} . $$

Later, in his thesis, [8], and reported separately with Avizienis, [7], he extended this analysis to the case of NMR/S, in which one has $N = 2n + 1$ and a majority, n + 1, of the basic N active units must be functional at all times. The main result was that

$$R(N,S)(T) = R^N R_s^S \left[1 + \sum_{j=0}^{S-2} \binom{NK+S}{j+1} \left(\frac{1}{R_s} - 1 \right)^{j+1} + \sum_{i=0}^{n} \binom{N}{i} \binom{NK+S}{S} \cdot \right.$$

$$\left. \cdot \sum_{r=0}^{i} \frac{\binom{i}{r} (-1)^{i-r}}{\binom{Kr+S}{S}} \left\{ \frac{1}{R_s^S R^r} - \sum_{j=0}^{S-2} \binom{Kr+S}{j+1} \left(\frac{1}{R_s} - 1 \right)^{j+1} \right\} \right] \qquad \text{Eq. 3}$$

In his thesis, Mathur also evaluated the MTBF (the Mean-Time-Before-Failure) of NMR/S systems, treating both the cases $\mu = 0$ and $\mu > 0$.

For $\mu = 0$, $S > 1$, he found:

$$\text{MTBF}(N,S) = \frac{1}{\lambda N} \left\{ 1 + (-1)^N \binom{2n}{n} + N^S \sum_{i=1}^{n} \binom{N}{i} \sum_{j=1}^{i} \binom{i}{j} (-1)^{i-j} \cdot \right.$$

$$\left. \cdot \left[\frac{1}{j^{S-1} (N-j)} - \sum_{r=1}^{S-1} \frac{1}{N^r j^{S-r}} \right] \right\} \qquad \text{Eq. 4a}$$

For $\mu = 0$, $S = 1$:

$$\text{MTBF}(N,1) = \frac{1}{\lambda N} + 1 + (-1)^n \binom{2n}{n} + \sum_{i=1}^{n} \binom{N}{i} \sum_{j=1}^{i} \binom{i}{j} \frac{(-1)^{i-j}}{N-j} \qquad \text{Eq. 4b}$$

Meanwhile, for $\mu > 0$, $S = 1$:

$$\text{MTBF}(N,S) = \mu \left\{ \sum_{m=0}^{S-1} \binom{NK+S}{m} \sum_{i=0}^{m} \binom{m}{i} \frac{(-1)^{m-i}}{NK+S-i} + \sum_{i=0}^{n} \binom{N}{i} \binom{NK+S}{S} \cdot \right.$$

$$\left. \cdot \sum_{r=0}^{i} \frac{\binom{i}{r} (-1)^{i-r}}{\binom{Kr+S}{S}} \left[\frac{1}{K(N-r)} - \sum_{m=0}^{S-1} \binom{Kr+s}{m} \cdot \sum_{i=0}^{m} \binom{m}{i} \frac{(-1)^{m-i}}{NK+S-i} \right] \right\}$$

Eq. 4c

Finally, for the case $\mu > 0$, $S = 1$,

$$\text{MTBF}(N,1) = \mu \left\{ \frac{1}{NK+1} + \sum_{i=0}^{n} \binom{N}{i} \sum_{r=0}^{i} \binom{i}{r} \frac{(-1)^{i-r}}{Kr+1} \left(\frac{NK+1}{NK-Kr} - 1 \right) \right\} \qquad \text{Eq. 4d}$$

The method he used to derive these quantities was principally that of enumeration of events combined with judicious integrations. The difficulty in attempting to employ this method in the present analysis is that it involves lengthy and complex decision trees, which become intricate to manipulate in dealing with the phenomena of fault detection and fault masking.

Along a somewhat different tack, Taylor [10] has treated a problem similar to TMR/S but he allows for software diagnostics to be added to the system when only two out of the original N + S modules are still in the non-failed state. Using a multiple integral evaluation routine, he computes the resultant module class reliability, which he calls $R^*_{TMR/S}$, and in which failure is declared only when all the units have failed. Letting $\tau = \lambda T$, he found that $R^*_{TMR/S}$ is given by:

$$\begin{aligned} R^*_{TMR/S} = e^{-3\tau} \Bigg\{ 1 &+ \sum_{n=1}^{N-3} \left[\prod_{i=0}^{n-1} \frac{(3+\mu i)}{(i+1)\mu} (1 - e^{-\mu\tau}) \right] \\ &+ \left[\prod_{n=0}^{N-3} \frac{(3+n\mu)}{\mu} \right] \left[\frac{(e^{\tau} - e^{-(N-3)\mu\tau})}{\prod_{n=0}^{N-3} \left(\frac{1+n\mu}{\mu} \right)} \right. \\ &\left. - \sum_{n=1}^{N-3} \frac{e^{-(N-3-n)\mu\tau} (1 - e^{-\mu\tau})^{n}}{n!} \prod_{i=0}^{N-3-n} \frac{\mu}{1+i\mu} \right] \\ &+ \left[\prod_{n=0}^{N-3} \left(\frac{3+n\mu}{\mu} \right) \right] \left[\frac{2(e^{2\tau} - e^{-(N-3)\mu\tau})}{\prod_{n=0}^{N-3} \left(\frac{2+n\mu}{\mu} \right)} - \frac{2(e^{\tau} - e^{-(N-3)\mu\tau})}{\prod_{n=0}^{N-3} \left(\frac{1+n\mu}{\mu} \right)} \right. \\ &\left. + 2 \sum_{n=1}^{N-3} \left(\frac{e^{-(N-3-n)\mu\tau} (1 - e^{-\mu\tau})^{n}}{n!} \cdot \left[\prod_{i=0}^{N-3-n} \left(\frac{\mu}{1+i\mu} \right) \cdot \prod_{i=0}^{N-3-n} \left(\frac{\mu}{2+i\mu} \right) \right] \right) \right] \Bigg\} \end{aligned} \qquad \text{Eq. 5}$$

Again, this approach does not lend itself readily to extension if one attempts to add the parameters of coverage. Bricker, in [1], completely unified the analyses of Kletsky, Mathur and Taylor by introducing the concept of hybrid degraded redundancy, written as H(N, S, D), in which one operates a system of $M = N + S$ units with N active and S spares until the total number of units falls to $D - 1$, $D \leq N$, at which point module class failure is declared. Kletsky's case is included by setting $D = N$, TMR/S as treated by Mathur is taken into account by setting $D = 2$, $N = 3$, while NMR/S is handled with $N = 2n + 1$, $D = n + 1$. Finally, Taylor's analysis corresponds to the case $N = 3$, $D = 1$. The method, using a combination of convolution of random variables and Laplace transform arguments, leads to the following simple expression for module class reliability:

$$R(N, S, D)(T) = \sum_{j=1}^{N+S-D+1} A_j^{N+S-D+1} e^{-\lambda_j T} \qquad \text{Eq. 6}$$

where

$$\lambda_j^i = \left\{ \begin{array}{ll} N\lambda + (S - (i-1))\mu & \text{if } 1 \leq i \leq S+1 \\ (N + S + 1 - i)\lambda & \text{if } S + 1 \leq i \leq N + S \end{array} \right.$$

and

$$A_j^i = \prod_{\substack{k \neq j \\ 1 \leq k \leq i}} \frac{\lambda_k}{(\lambda_k - \lambda_j)}$$

For computational purposes this result is considerably more efficient to use as well as being more general than the results given by equations 2, 3 and 5.

It should also be noted that as an immediate consequence of this approach the module class MTBF is revealed by inspection to be:

$$\text{MTBF}\left\{H(N, S, D) \text{ system}\right\} = \sum_{k=0}^{S} \frac{1}{\lambda N + k\mu} + \sum_{k=D}^{N-1} \frac{1}{\lambda k} \quad \text{for } \mu > 0 \qquad \text{Eq. 7a}$$

$$= \sum_{k=D}^{N-1} \frac{1}{\lambda k} + \frac{S+1}{\lambda N} \quad \text{for } \mu = 0 \qquad \text{Eq. 7b}$$

Setting D = n + 1, we obtain the MTBF of NMR/S systems in a simpler form than that provided by Eqs 4a-4b. Setting D = 1, N = 3, we obtain the MTBF of Taylor's (TMR/S)* systems.

However, despite the ease and straightforwardness of the analysis provided in [1] the method is not readily generalizable in its approach, to treat the complexities of imperfect fault-detection systems which will be examined in II. B, below.

B. The Case of Imperfect Coverage

The first major contribution to the problem of imperfect fault-tolerant computer reliability, using error detection and correction type implementation is generally attributed to J. P. Roth, et. al. in [2], [3], [4]. Roth defined the "coverage" parameter c to be:

$$c = \text{Probability (System recovers} \mid \text{a module failure has occurred)}$$

For ARMMS purposes the system in question is the module class. The authors go on to state:

"Exactly what constitutes recovery is a matter for the individual system designer to settle; at this point it is just a system parameter. In some situations recovery may only mean detection, location and automatic repair of the hardware failure, while in others it may also include very complex restoration of an operating data base. In a sense, c can be interpreted as a probability of surviving a failure without irreparable damage."

Mathur, in his thesis, P. 34-35, concurs with Roth, et. al., in their treatment of the fault detection problem as exemplified by the introduction of the coverage factor. He writes, in discussing a previous model by Flehinger, which treated the detailed behavior of the switching mechanism in a standby-replacement system which she investigated:

"The actual switching mechanisms utilized, the error detection codes, along with code check circuitry, and the software requirements of program rollback are very much implementation dependent. Any detailed modeling of these effects would necessarily be constrained to a narrow range of implementation possibilities. Hence, Bouricius, Roth, et. al., in [3], avoiding the multifarious parameters as exemplified by the Flehinger model, conceived a single parameter, c, which takes into account all the aspects of failure detection and recovery. Thus, the exact definition of how in both the active and passive states. Since the occurrence of faults is modeled in terms of hazard rates one has the following density functions for the occurrences of maskable active, unmaskable active, maskable passive and unmaskable passive faults for a given module, (where unmaskable connotes detectable as well, implicitly) $\lambda_m e^{-\lambda_m t}$, $\lambda_{\bar{m}} e^{-\lambda_{\bar{m}} t}$, $\mu_m e^{-\mu_m t}$, $\mu_{\bar{m}} e^{-\mu_{\bar{m}} t}$. Then the coverage relating to the maskable active and unmaskable states would clearly be given by $c_{ma} = P_m = \lambda_m/\lambda$ and $c_{\bar{m}a} = P_{\bar{m}} = \lambda_{\bar{m}}/\lambda$. However, the probability that a module class would recover, given that a fault has occurred to a spare, must now be conditioned on the number of faults of each kind that have occurred to each spare, and this depends on time as well. For example, given only one spare left and the occurrence of a fault in that spare when the module class is operating in simplex at time t of a mission of total duration T, then the coverage relating to a maskable fault would be given by:

$$\int_0^{T-t} \frac{d}{ds}\left[1 - (1 + \lambda_m s)\, e^{-(\lambda_m + \lambda_{\bar{m}})s}\right] e^{-(\lambda - \lambda_m - \lambda_{\bar{m}})s} e^{-\mu s} ds \qquad \text{Eq. 8}$$

$= \text{A function of } T-t$

This would be the case if the active unit has experienced no faults at the instant that the spare had acquired the maskable fault. A similar expression would hold in the case that the active module had already acquired one maskable fault.

Let us now turn to the basic Roth coverage model as described in [3] and [4], to evaluate ${}^{0}_{c}R^{N}_{S}(\lambda, \mu; T)$. This analysis involved the method of recursive integral equations.

We let ${}_{c}R^{N}_{S}(T) = {}^{0}_{c}R^{N}_{S}(\lambda, \mu; T)$, for convenience, since the parameters λ, μ will be held constant throughout. Then Roth, et. al., obtained the basic integral equation:

$${}_{c}R^{N}_{S}(T) = {}_{c}R^{N}_{S-1}(T) + c\int_0^T \frac{d}{dt}\left[1 - cR^{N}_{S-1}(t)\right] e^{-\mu t} e^{-\lambda(T-t)} dt \qquad \text{Eq. 9a}$$

with initial condition given by:

$${}_{c}R^{N}_{0}(T) = e^{-\lambda T}$$

This was solved to yield,

$${}_{c}R^{N}_{S}(T) = \sum_{i=0}^{S} \binom{S}{i} c^{i} (1-c)^{S-i} R_i(T) \qquad \text{Eq. 9b}$$

the system failure is detected, how the switching of spares is to be implemented and what constitutes recovery has to be answered by the system designers of the particular system under scrutiny. For the purposes of reliability modeling these variants are lumped into one variable, the coverage factor c".

The point of view of the present study (and also that of some recent work of Rennels & Avizienis, [13], [14]) is that both Roth, et. al. and Mathur have presented an oversimplified view of the coverage concept and that lumping the factors together into a c factor is misleading. In fact, it is felt by the present author that a model, following more closely along the lines of the Flehinger model, referred to above, which attempts to delineate the important components of fault detection and correction, is the proper one to emulate to the extent that it is mathematically possible. In the ARMMS reliability model, the factors of fault detection and masking are explicitly stated and it is not readily discernible how a single factor c could incorporate these basic features inherent in the coverage mechanism. Thus, the present analysis attempts to embody coverage in terms of a vector of components $\vec{c} = (\lambda_d, \lambda_m, \mu_d, \mu_m)$ where (λ_d, λ_m) and (μ_d, μ_m) are the components of the hazard rates λ and μ, respectively, relating to detectable and maskable type faults in the active and passive states, respectively.

Although failures which occurred in the passive mode would be neither detectable nor maskable at the moment that they occurred, at the instant of switchover it is assumed in our model that they would be detected and masked (in the multiprocessor case, e.g.) with respective probabilities given by $P'_d = \mu_d/\mu$ and $P'_m = \mu_m/\mu$, due to the availability of software diagnostic routines that would test these modules in a duplex mode prior to using them in a multiprocessor mode. Moreover, this vector will actually change when the module class reverts to the Duplex or TMR modes. This complexity of operation in three or more distinct modes as well as the refinements required to distinguish the active from the passive states, and, in addition, the distinction to be made between fault detection and fault masking precludes the possibility of adapting the Roth coverage concept to ARMMS reliability requirements.

To examine more closely the distinction to be made between our use of coverage and that of Roth, et. al., note first that for Roth c is a constant independent of whether a fault in a module occurred in the active or passive state. In the coverage model for ARMMS one distinguishes between maskable v. s. unmaskable coverage where $R_i(T)$ is given by ${}_1^0R_S^N(\lambda,\mu;T)$ from Eq. 1.

However, this formulation is incorrect since a fault in the system consisting of S - 1 spares and N working units due to an error in detection may not allow for the use of the last spare, a fact ignored in the inclusion of the term $d/dt\left(1 - {}_cR_{S-1}^N(t)\right)$ in Eq. 9'a, above.

This error negates all of the coverage computations in [3] and [4]. However, the authors rectified their error in [2], which was published two years later, and here they gave the correct formulation of ${}_c^0R_S^N(T)$ as follows:

$${}_c^0R_S^N(T) = {}_cR_S^N(T) = {}_c^0R_{S-1}^N(T) + c^S \cdot \int_0^T \frac{d}{dt}\left[1 - {}_1R_{S-1}^N(t)\right] e^{-\mu t} e^{-N\lambda(T-t)}dt \qquad \text{Eq. 9a}$$

with initial condition ${}_cR_0^N(T) = e^{-N\lambda T}$.

The solution of Eq. 9a is easily derived by the following algebraic argument:

$${}_cR_S^N = {}_cR_{S-1}^N + c^S \int_0^T \frac{d}{dt}(1 - {}_1R_{S-1}^N) e^{-\mu t} e^{-N\lambda(T-t)}dt \qquad \text{Eq. 9b}$$

But setting c = 1 gives rise to the case given by Eq. 1, i.e.,

$${}_1R_S^N = {}_1R_{S-1}^N + \int_0^T \frac{d}{dt}(1 - {}_1R_{S-1}^N) e^{-\mu t} e^{-N\lambda(T-t)}dt \qquad \text{Eq. 9c}$$

Thus

$$\int_0^T \frac{d}{dt}(1 - {}_1R_{S-1}^N) e^{-\mu t} e^{-N\lambda(T-t)}dt = {}_1R_S^N - {}_1R_{S-1}^N \qquad \text{Eq. 9d}$$

and substituting ${}_1R_S^N - {}_1R_{S-1}^N$ for the integral appearing in (9b) yields,

$${}_cR_S^N = {}_cR_{S-1}^N + c^S({}_1R_S^N - {}_1R_{S-1}^N) \qquad \text{Eq. 9e}$$

Thus, by recursion, we have

$${}_cR_{S-1}^N = {}_cR_{S-2}^N + c^{S-1}({}_1R_{S-1}^N - {}_1R_{S-2}^N) \qquad \text{Eq. 9f}$$

$${}_cR_{S-2}^N = {}_cR_{S-3}^N + c^{S-2}({}_1R_{S-2}^N - {}_1R_{S-3}^N) \qquad \text{Eq. 9g}$$

$${}_cR_1^N = {}_cR_0^N + c({}_1R_1^N - {}_1R_0^N) \qquad \text{Eq. 9h}$$

Summing in equations 9b→9h yields:

$${}_cR_S^N = \sum_{k=0}^{S} c^k ({}_1R_k^N - {}_1R_{k-1}^N)$$

where

$${}_1R_{-1}^N = 0$$

Using Eq. 1 and observing that $\binom{k-1+KN}{k} \cdot (1 - e^{-\mu T}) = {}_1R_k^N - {}_1R_{k-1}^N$, we finally arrive at the conclusion:

$${}_cR_S^N = e^{-N\lambda T} \sum_{k=0}^{S} c^k \binom{k-1+NK}{k} \left(1 - e^{-\mu T}\right)^k \qquad \text{Eq. 10}$$

A derivation of the MTBF for the Roth-Bouricius coverage model may also be obtained algebraically from the integral equation formulation. In fact, from Eq. 9e, above, we have:

$$\int_0^\infty {}_cR_S^N(t)\,dt = c^S\left[\int_0^\infty {}_1R_S^N(t)\,dt - \int_0^\infty {}_1R_{S-1}^N(t)\,dt\right] + \int_0^\infty {}_cR_{N-1}^S(t)\,dt \qquad \text{Eq. 10a}$$

Letting ${}_cE_S^N$ = MTBF of the module class system when the coverage is equal to c, we find that

$${}_cE_S^N = \int_0^\infty {}_cR_S^N(t)\,dt = c^S({}_1E_S^N - {}_1E_{S-1}^N) + {}_cE_{S-1}^N \qquad \text{Eq. 10b}$$

and, in general, for $1 \leq k \leq S$,

$${}_cE_k^N = {}_cE_{k-1}^N + c^k\left({}_1E_k^N - {}_1E_{k-1}^N\right) \qquad \text{Eq. 10c}$$

Since, from Eq. 7a, setting D = N, the MTBF of a parallel N-active, k-spare module class system with perfect coverage is given by:

${}_1E_k^N + \sum_{r=0}^{k} \frac{1}{\lambda N + r\mu}$, it follows that,

$${}_1E_k^N - {}_1E_{k-1}^N = \frac{1}{\lambda N + k\mu}, \qquad \text{Eq. 10d}$$

Then, summing in Eq. 10c for k = 1, 2, . . . S yields,

$${}_{c}E_{S}^{N} = \sum_{k=1}^{S} \frac{c^{k}}{\lambda N + k\mu} + {}_{1}E_{0}^{N} = \sum_{k=0}^{S} \frac{c^{k}}{\lambda N + k\mu} \quad . \qquad \text{Eq. 10e}$$

In summary, in regard to the Roth-Bouricius reliability model, no distinction is made between fault detection and fault masking and every fault is assumed to cause the module which sustains that fault to be removed from the system. For ARMMS these assumptions aren't tenable and thus the Roth-Bouricius model cannot be employed for this system.

In [2], Bouricius also treated the special case $\lambda = \mu$ to obtain ${}^{f}_{c}R_{S}^{1}(\lambda, \lambda; T)$, for arbitrary f. His equations were obtained by case enumeration:

$${}^{f}_{c}R_{S}^{1}(\lambda,\lambda;T) = {}^{f}_{c}R \sum_{i=0}^{S} \left[c^{f+1}(1 - {}^{f}R)\right]^{i} \qquad \text{Eq. 11}$$

where

$${}^{f}_{c}R = {}^{f}_{c}R_{0}^{1}(T) = e^{-\lambda T} \sum_{k=0}^{f} \frac{(c\lambda T)^{k}}{k!} \qquad \text{Eq. 11a}$$

and

$${}^{f}R = {}^{f}_{1}R_{0}^{1}(T) = e^{-\lambda T} \sum_{k=0}^{f} \frac{(\lambda T)^{k}}{k!} \qquad \text{Eq. 11b}$$

Although this would be a useful result and could serve as a springboard for studying the evaluation of simplex reliability for ARMMS memory modules (by considering the special case f = 1, $\lambda = \mu$ and $P_m = P_d$) it turns out that the argument required to justify Eqs. 11-11b is in error. The error stems from the fact that accumulated failures for an active unit are detected sequentially in time, whereas, when the system is ready to switch in a spare module, this spare may already have acquired a set of faults which wouldn't have been detected while the module was in the dormant state. In particular, the term $c^{f+1}(1 - {}^{f}R)$ is supposed to represent the probability that a unit will have acquired at least f + 1 faults in time [0, T], and that at least f + 1 of these faults were detectable, so that the unit would be discarded and the module class would not be declared to have failed. This would indeed be the case for the first unit to be used actively in the system, since faults occur sequentially in time in regard to this unit. However, for the 2nd, 3rd, . . . etc. modules which were initially in the dormant (spare) mode, this no longer holds. Thus, for a spare which as acquired f + k faults, with $k \geq 1$, by the time one is ready to implement its use in the system, the term c^{f+1} no longer represents the probability that some subset of f + 1 of these f + k faults was detectable. To determine the appropriate value of ${}^{f}_{c}R^{1}_{S}(\lambda, \lambda; T)$ it would be necessary to examine the time interval [0, T], locally, and distinguish between the number of faults which a spare module has sustained prior to its on-line switch-over into the system.

To do this it would be necessary to define coverage, c, in a more general way, e.g., in terms of the number of faults, r, which a module has sustained, where r need no longer be equal to one:

$$c_r = \text{Prob}\left\{\text{System recovers} / \text{r faults have occurred}\right\}$$

It would appear from these remarks and some observations due to Rennels & Avizienis ([13], [14]) that it is essential to distinguish coverage in cases of multiple faults and in regard to faults occurring in the dormant v. s. the active mode. Thus the miscalculations inherent in Eqs. 11-11b would appear to point up the requirement for a more critical general definition of coverage. As Roth-Bouricius and Mathur have themselves clearly indicated, coverage is design and implementation dependent and the definition must relate to specific design features such as fault masking and fault detection; although they conceded the first point, they thought the second point could be neglected.

It should be noted, however, that Eqs. 11-11b do hold in the special case $f = 0$; in fact, setting $f = 0$, $\lambda = \mu$ and $N = 1$ in Eq. 10 above, yields Eq. 11. This is due to the fact that the definition of coverage, c, is really quite different in the interpretation for $f = 0$ versus that used for $f > 0$.

When

$$\begin{aligned} f = 0, \quad c &= \text{Prob (System recovers/system failure)} \\ &= \text{Prob (System recovers/module failure)} \end{aligned}$$

since a dormant module failure requires no system recovery at all until one is ready to switch in the dormant module.

When

$f = 0$,

then,

c = Prob (System recovery/At least one fault has occurred in some module)

Now let us uncritically accept the assumption that c is independent of the actual number of faults that have occurred in a module, so that one has the same chance of detecting the fact that a module has failed whether the failure is due to 1 fault or to 10 faults. The Hamming error-detecting codes would make this particular assumption invalid but we shall assume that it would be possible to invent some design of an error detection mechanism that would validate this claim (i.e., the interpretation of c may be unrealistic in terms of engineering design but it represents no mathematical impossibility). Then, in examining the derivation of Eqs. 11-11b, no difficulty ensues, so that Eq. 10 and Eq. 11 are equivalent when $f = 0$.

Let us now turn to the case $f > 0$. In order to obtain the factor c^{f+1}, in the case of dormant fault occurrences, one must define c as follows:

c = Prob (Module fault can be detected/module fault has occurred)

otherwise, using the previous definition one would simply have the term $c(1 - {}^{f}R)^{i}$ instead of $c^{f+1}(1 - {}^{f}R)^{i}$ in Eq. 11. But with this interpretation the temporal sequence of faults must be considered before Eqs. 11-11b can be rectified. Clearly the ubiquitous constant c must be carefully examined depending on its context of application.

Another attempt at developing an analytic approach to the coverage problem was performed by Wyle and Burnett in [12]. The underlying system is of the Kletsky-type i.e., an N-active, S-spare module class system under the following additional assumptions:

a) $\lambda = \mu$

b) No off line spares, or equivalently, any undetected failure in a power-off or power-on state causes a module class failure

c) No fault masking, i.e., $f = 0$.

The authors derived the reliability:

$$R(T) = 1 - \sum_{k=N}^{N+S} \binom{N+S}{k} P_f^k (1 - P_f)^{N+S-k}$$

$$- \sum_{k=1}^{S} \binom{N+S}{k} P_f^k (1 - P_f)^{N+S-k} (1 - c^k) \qquad \text{Eq. 12}$$

where

$$P_f = 1 - e^{-\lambda T}$$

and

c is coverage in the Roth-Bouricius sense.

This model has serious shortcomings in assumptions b) and c) above.

A model with considerably greater depth was provided by Rennels & Avizienis in [13]. They describe a model for coverage in standby redundant systems, which in our hybrid notation are describable as H(N, S, N) type systems, involving two essential parameters A_c^a and A_c^s where:

1) A_c^a represents the conditional probability that a properly functioning monitor unit can effect recovery, given that a fault occurs in one of the N active modules.

2) A_c^s represents the conditional probability that a properly functioning monitor unit can effect recovery, given that one or more faults have occurred in a spare unit and show up when it is activated.

(The monitor unit is analogous to BOSS in the ARMMS system.)

This refinement of the single parameter coverage concept is further developed by the authors in the context of f = 0, i. e., no failures or faults are to be tolerated per module. Let $\vec{c}$ represent the vector (A_c^a, A_c^s); then they obtain the recursive formulation

$$_{\vec{c}}R_S^N(T) = e^{-N\lambda T} e^{-S\mu T} + (A_c^a \lambda N + A_c^s S\mu) \int_0^T e^{-(N\lambda + S\mu)x} {}_{\vec{c}}R_{S-1}^N (T-x)dx$$

$$+ \sum_{i=1}^{S} (1 - A_c^s \mu) \int_0^T e^{-(N\lambda + S\mu)x} {}_{\vec{c}}R_{i-1}^N (T-x)dx \qquad \text{Eq. 13a}$$

with initial condition,

$$_{\vec{c}}R_S^N(T) = e^{-N\lambda T}$$

These equations are solved recursively via the schema:

$$_{c}R_{S}^{N}(T) = e^{-N\lambda T} \sum_{i=0}^{S} A_{S,i}\, e^{-i\mu T} \qquad \text{Eq. 13b}$$

where

$$A_{S,i} = \frac{(A_c^a NK + S A_c^s)\, A_{S-1,i} + (1 - A_c^s) \sum_{0=i}^{S-1} A_{j,i}}{S - i} \quad \text{for } S > i$$

Eq. 13c

$$A_{S,S} = 1 - \sum_{i=0}^{S-1} A_{S,i};\ A_{0,0} = 1;\ A_{i,j} = 0 \text{ for } j > i$$

This brief summary of the coverage reliability problem has indicated that very few significant studies have been made, and when confronted with a specific fault tolerant design such as ARMMS, it is not surprising that no tailor made analyses already exist for one's use. In the next section we shall describe a mathematical model which was developed explicitly for the ARMMS coverage problem. The model is algorithmic (as opposed to being of the Monte Carlo or simulation variety) but there are no general closed form answers to the equations developed, and numerical programming procedures are required in order to evaluate the multiprocessor, Duplex and TMR reliability performances.

III. Mathematical Formulation of Simplex, Multiprocessor, Duplex and TMR Reliability Including Fault Detection and Fault Masking

III. A. Model Assumptions for Simplex and Multiprocessor Modes

A-1 - Only one failure can be masked per module. Additional maskable faults will be detectable but the module will be removed from on-line and not used in this mode again.

A-2 - Any number of undetectable faults per module will remain undetectable and any number of detectable faults may occur per module and still remain detectable as a group. (This assumption is not strictly true for ARMMS but for the range of hazard rates anticipated in the program, the exceptions may be considered negligible for reliability modeling purposes).

A-3 - Faults that cannot be detected in simplex for a module that developed faults while dormant can be detected in duplex with probability $P_d = 1$ and in the case of processors one may assume that a processor can be tested in duplex prior to placing it on line in simplex. (At present the model has been structured without this assumption since it is more difficult to add diagnostic subroutines. Also the analysis is more complex for this case.)

A-4 - In the case of memory modules it is to be assumed that $\lambda = \mu$.

A-5 - Serial Gate Model Assumption:

Let

g	= # of gates/module
g_u	= # of gates, the failure of any one of which would cause an undetectable module failure
g_d	= # of gates, the failure of any one of which would cause a detectable module failure
g_m	= # of gates, the failure of any one of which would cause a maskable module failure
$g_{\overline{m}}$	= # of gates, the failure of any one of which would cause an unmaskable but detectable module failure

Then,

$$g = g_u + g_d = g_u + g_m + g_{\overline{m}} \qquad \text{Eq. 14a}$$

$$\lambda = \lambda_u + \lambda_d = \lambda_u + \lambda_m + \lambda_{\overline{m}} \qquad \text{Eq. 14b}$$

Eq. 14b merely restates Eq. 14a in terms of the failure rates associated with the detectable, maskable, etc. portions of the module hardware.

We suppose that all gates are in series. Furthermore, we consider the ratio of the hazard rate for second failure to the hazard rate for first failure for the three basic failure types U, M and $\overline{M}$ in Table 1, following.

For large values of g_u, g_m, $g_{\overline{m}}$, it follows that Table 1 has approximately the entries that it would have if the second failures were independent of the first. Moreover, if λ is sufficiently small, the chance of an appreciable number of failures is small so that one has the basic Serial Module Gate Assumption, viz.,

1 \ 2	U	M	$\overline{M}$
U	$\lambda_u - 1/g$	λ_m/λ	$\lambda_{\overline{m}}/\lambda$
M	λ_u/λ	$\lambda_m/\lambda - 1/g$	$\lambda_{\overline{m}}/\lambda$
$\overline{M}$	λ_u/λ	λ_m/λ	$\lambda_{\overline{m}}/\lambda - 1/g$

Table 1 - Ratio of Hazard Rate to Second Failure to Hazard Rate to First Failure for the Failure Types U, M, $\overline{M}$

repeated failures are independent and are generated as negative exponential random variables with parameters λ, λ_d, λ_u, λ_m, $\lambda_{\overline{m}}$ for the respective types of gate hardware. Clearly,

$$P_d = \frac{\lambda_d}{\lambda}, \quad P_m = \frac{\lambda_m}{\lambda}, \quad P_{\overline{m}} = \frac{\lambda_{\overline{m}}}{\lambda},$$

when the module is operated in the active mode.

II. B. Glossary of Symbols

$\frac{1}{c}R_S^1(\lambda, \mu; T)$	=	Prob (Successful simplex operation in $[0, T]$ with S available spares, λ, μ the hazard rates in the active and passive modes, while the coverage vector is $\vec{c}$)
$\frac{1}{c}R_S^N(\lambda, \mu, T)$	=	Prob (Successful multiprocessor operation in $[0, T]$ with S available spares, λ, μ the hazard rates in the active and passive modes, while the coverage vector is $\vec{c}$)
$\vec{c}$	=	The coverage vector, given by $(\lambda_m, \lambda_{\overline{m}}, \mu_m, \mu_{\overline{m}})$
λ	=	Active hazard rate
μ	=	Passive hazard rate
λ_m	=	Active hazard rate for maskable error hardware
μ_m	=	Passive hazard rate for maskable error hardware
$\lambda_{\overline{m}}$	=	Active hazard rate for unmaskable error hardware

$\mu_{\overline{m}}$	=	Passive hazard rate for unmaskable error hardware
M	=	# of available modules in the module class at time 0
f	=	# of all allowable maskable faults per module (for the current analysis f = 1)
P_d	=	Prob (Detection occurs $\mid$ module failure in the active mode) = $\frac{\lambda d}{\lambda}$
P_m	=	Prob (Detection and fault masking occurs $\mid$ module failure occurs in the active mode) = $\frac{\lambda m}{\lambda}$

III. C. The Birth-Death Process Analysis for Simplex Reliability

II. C. 1. General Discussion

The major complexity in dealing with the fault-detection and correction problem in the present model lies in the fact that when an unpowered spare is powered on the unit changes its hazard rate from μ to λ and this presents combinatorial as well as analytic difficulties. In addition, one must keep track of the order of events in which transitions of this type are occurring since different probabilities are to be attached to differing transition types. A basic method for taking account of the transitions from the passive to the active mode is that of the Birth and Death Process, since due to the negative exponential character of the various modules in the active and passive states, one has an underlying Markov process in effect.

We assume that the units are to be run sequentially and that starting with unit 1 in the active state, we operate it until it has accumulated one non-maskable fault or two maskable faults, whichever event occurs first. At that point in time we switch over to the next passive unit in sequence which has the property that it has either acquired no unmaskable faults or at most one maskable fault, this having been acquired while in the passive state, and power this unit on. This unit is then operated in a manner identical to that in which the first unit was operated and one proceeds in sequence through the entire bank of S + 1 = M modules. Module class failure occurs if before time T some module was actively run with an undefected fault or there are no modules left among the M with the property that at most one maskable fault has been acquired by that module.

We shall not invoke Assumption A-3 here, but rather will take the point of view that if an unpowered unit is to be placed on line, then an undetected failure will have occurred with hazard rate $\mu - \mu_d$ during the period that the unit was in the dormant state. The corresponding passive maskable and unmaskable hazard rates are given by μ_m and $\mu_{\overline{m}}$, respectively, where $\mu_d = \mu_m + \mu_{\overline{m}}$. Again we think in terms of the gate model assumption, A-5, and that faults of the three different types, undetectable, maskable and unmaskable may be conceived of in terms of three different hardware classes. Also, A-5 implied that successive faults were independently distributed (approximately) both with respect to a given hardware type and with respect to different hardware types, (as in Table 1, p. 18), except that the λ-symbols should be replaced with their μ-counterparts.

In Figure 1, below, the general inclusion relations that exist among the various fault types are displayed

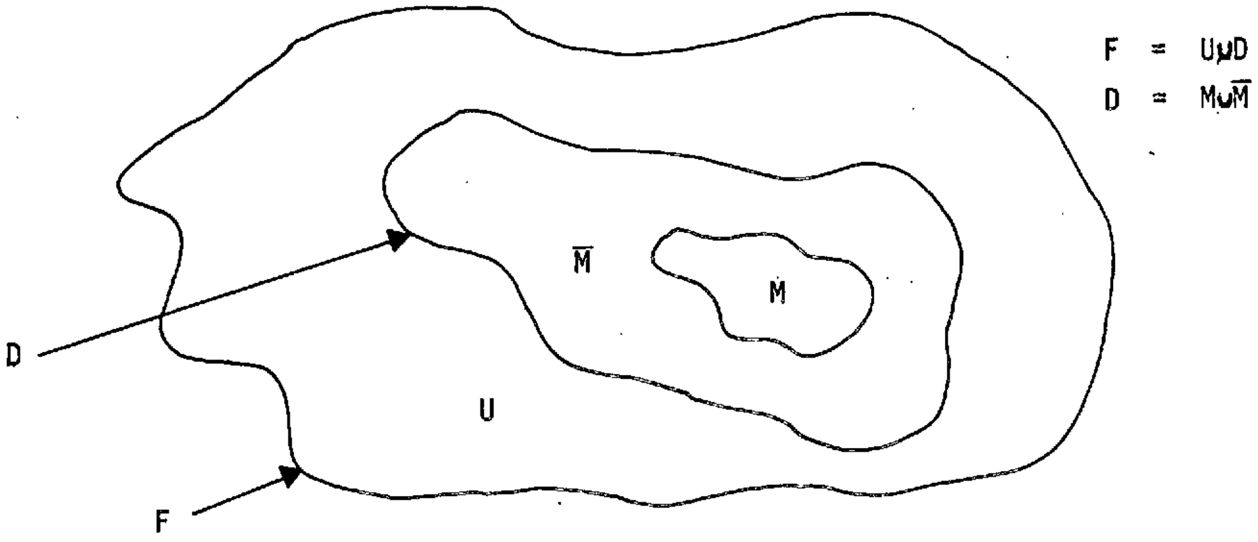


Figure 1. Fault Type Inclusion Relationships

U = Class of all undetectable faults

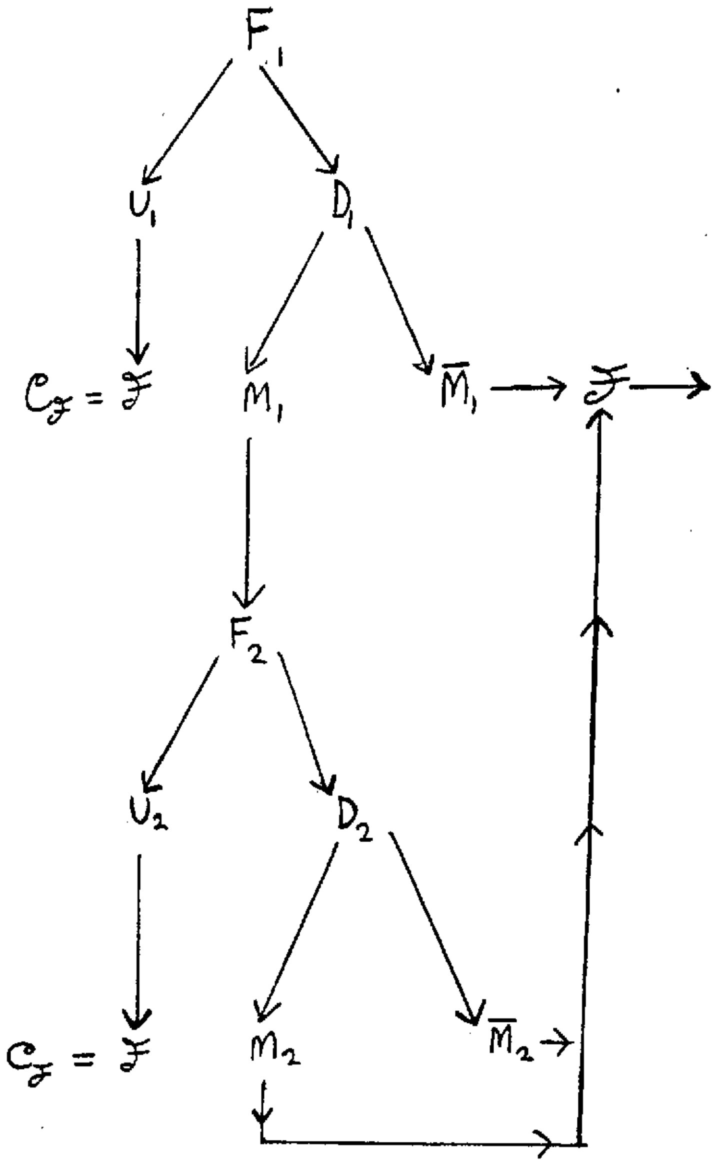
M = Class of all maskable faults

$\overline{M}$ = Class of all unmaskable faults

F = Class of all faults

D = Class of all detectable faults

A simple decision tree illustrates the typical event sequences associated with a fault; this is given in Figure 2, below, and applies to active faults only.



F_1
U_1
D_1
$\mathcal{C}_{\mathfrak{F}} = \mathfrak{F}$
M_1
$\overline{M}_1$
$\mathfrak{F}$
F_2
U_2
D_2
$\mathcal{C}_{\mathfrak{F}} = \mathfrak{F}$
M_2
$\overline{M}_2$

Go on to next module in sequence and power it on.

$\mathfrak{F}$ refers to module failure.

$\mathcal{C}_{\mathfrak{F}}$ refers to module class failure.

Figure 2. A Flow Diagram Depicting a Failure Sequence for an Active Module

In Figure 2, above, the subscripts 1 and 2 refer to first and second fault occurrences, respectively. For an inactive module, faults cannot be detected nor masked until the unit is powered on, at which point the fault will be detected with probability $\frac{\mu_d}{\mu}$ instantaneously if a fault occurred and will be masked, instantaneously, with probability $\frac{\mu_m}{\mu}$ if a fault occurred. The instantaneous time assumption is reasonable within the framework of presently structured ARMMS hardware design.

II. C. 2. The State Space and Differential Equations of the Simplex Reliability Analysis

At time t the system (module class) will be said to be in state (i, j) if the module class hasn't failed in the interval [0, t] and if module i is active at time t and has experienced j maskable faults and no unmaskable faults. It is implicit that if the module class hasn't failed in [0, t] that the module numbered (indexed) by i has experienced no undetectable faults during [0, t].

Here,

$1 \le i \le M, \quad j = 0, 1$

Let

$P_{i,0}(t) = \text{Prob (System is in state (i, 0) at time t)}$

$P_{i,1}(t) = \text{Prob (System is in state (i, 1) at time t)}$

Then

$$\frac{1}{c} R_S^1 (\lambda, \mu; T) = \sum_{i=1}^{M} \left[P_{i,0}(t) + P_{i,1}(t)\right] \qquad \text{Eq. 15}$$

Furthermore, for i ≥ 2 we have the transition equations:

$$P_{i,0}(t+\Delta t) = \sum_{j=1}^{i-1} P_{j,0}(t) \left[1 - (1+\mu_m t) e^{-\mu_d t}\right]^{i-j-1} \lambda_{\overline{m}} e^{-\mu t} \Delta t \qquad \text{Eq. 15a}$$

$$+ \sum_{j=1}^{i-1} P_{j,1}(t) \left[1 - (1+\mu_m t) e^{-\mu_d t}\right]^{i-j-1} \lambda_d e^{-\mu t} \Delta t$$

$$+ P_{i,0}(t) \left[1 - \lambda \Delta t\right] + o(\Delta t)$$

$$P_{i,1}(t + \Delta t) = \sum_{j=1}^{i-1} P_{j,0}(t) \left[1 - (1 + \mu_m t)e^{-\mu_d t}\right]^{i-j-1} \lambda_{\overline{m}} \mu_m t\, e^{-\mu t} \Delta t \qquad \text{Eq. 15b}$$

$$+ \sum_{j=1}^{i-1} P_{j,0}(t) \left[1 - (1 + \mu_m t)e^{-\mu_d t}\right]^{i-j-1} \lambda_d \mu_m t e^{-\mu t} \Delta t$$

$$+ P_{i,0}(t) (\lambda_m \Delta t) + P_{i,1}(t) [1 - \lambda \Delta t] + o(\Delta t)$$

For $i = 1$,

$$P_{1,0}(t + \Delta t) = P_{1,0}(t) [1 - \lambda \Delta t] + o(\Delta t) \qquad \text{Eq. 15c}$$

$$P_{1,1}(t + \Delta t) = P_{1,0}(t) [\lambda_m \Delta t] + P_{1,1}(t) [1 - \lambda \Delta t] + o(\Delta t)$$

Equation 15a is derived as follows:

In order to be in state $(i, 0)$ at time $t + \Delta t$ it is necessary that either,

a) The state at time t was $(i, 0)$, the probability of this event being $P_{i,0}(t)$ and in the duration of time Δt the Poisson process of faults related to module i had no arrivals, i.e., $1 - \lambda \Delta t + o(\Delta t)$ = probability that no failure occurred, or

b) At t the system was in state $(j, 0)$, for $j<i$, module j experienced a non-maskable fault, with probability given by $\lambda_{\overline{m}} \Delta t + o(\Delta t)$, each of the i-j-1 modules of index l, where $i<l<j$, had either at least two maskable faults or at least one non-maskable faults in the powered-off state, this probability being given by $[1 - (1 + \mu_m t)e^{-\mu_m t} e^{-\mu_{\overline{m}} t}]$ and module i had no faults in the internal $(0,t)$, this occurring with probability $= e^{-\mu t}$, or

c) At time t the system was in state $(j, 1)$ for $j<i$, with probability $= P_{j,1}(t)$ and furthermore, each of the intervening j-i-1 modules experienced at least two maskable or one non-maskable fault, and module j experienced a detectable fault with probability $\lambda_d \Delta t + o(\Delta t)$, while, finally, module i had no faults in $[0, t]$ nor in $[t, t + \Delta t]$.

A similar derivation holds for Equation 15b with the factor $e^{-\mu t} (\mu_m t)$ representing the probability that module i acquired exactly one maskable fault in the time interval $[0, t]$.

In Equations 15a, b, c, and d the term $o(\Delta t)$ denotes an error term which satisfies $\frac{o(\Delta t)}{\Delta t} \rightarrow 0$ as $\Delta t \rightarrow 0$.

The Equations 15a, b depict a system satisfying the assumptions of A1→A5, except that duplex testing for faults isn't performed for modules moving from the power-off to the power-on state.

Subtracting $P_{i,0}(t)$ and $P_{i,1}(t)$ from the right-hand sides of Equations 15a, b, respectively, and then dividing both sides by Δt and letting $\Delta t \rightarrow 0$, one obtains the following system of differential difference equations:

For $i \geq 2$,

$$\dot{P}_{i,0}(t) = \lambda P_{i,0}(t) + \sum_{j=1}^{i-1} P_{j,0}(t) \left[1 - (1 + \mu_m t)e^{-\mu_d t}\right]^{i-j-1} \lambda_{\overline{m}} e^{-\mu t} \qquad \text{Eq. 16a}$$

$$+ \sum_{j=1}^{i-1} P_{j,1}(t) \left[1 - (1 + \mu_m t)e^{-\mu_d t}\right]^{i-j-1} \lambda_d e^{-\mu t}$$

$$\dot{P}_{i,1}(t) = \lambda_m P_{i,0}(t) - \lambda P_{i,1}(t)$$

$$+ \sum_{j=1}^{i-1} P_{j,0}(t) \left[1 - (1 + \mu_m t)e^{-\mu_d t}\right]^{i-j-1} \lambda_{\overline{m}} \mu_m t\, e^{-\mu t}$$

$$+ \sum_{j=1}^{i-1} P_{j,1}(t) \left[1 - (1 + \mu_m t)e^{-\mu_d t}\right]^{i-j-1} \lambda_d \mu_m t\, e^{-\mu t}$$

For $i = 1$

$$\dot{P}_{1,0}(t) = -\lambda P_{1,0}(t) \qquad \text{Eq. 15c}$$

$$\dot{P}_{1,1}(t) = -\lambda P_{1,1}(t) + \lambda_m P_{1,0}(t) \qquad \text{Eq. 15d}$$

The initial conditions are given by

$$P_{1,0}(0) = 1 \ , \quad P_{1,1}(0) = 0 \qquad \text{Eq. 15e}$$

$$P_{i,0}(0) = 0 \ , \quad P_{i,1}(0) = 0 \quad \text{for } i \geq 2 \qquad \text{Eq. 15f}$$

For relatively small i, it is possible to solve the system of differential equations 15a, b, c, d subject to the initial conditions 16a, b directly. For larger i, a computer algorithm is necessary. It should be noted that $P_{i,k}(t)$ depends only upon $P_{j,1}(t)$ for $\substack{j \le i \\ l \le k}$, k = 0, 1, all t.

For i = 1 we find:

$$P_{1,0}(t) = e^{-\lambda t} \qquad \text{Eq. 17a}$$

$$\dot{P}_{1,1}(t) = -\lambda P_{1,1}(t) + \lambda_m e^{-\lambda t}$$

$$P_{1,1}(t) = \lambda_m t\, e^{-\lambda t} \qquad \text{Eq. 17b}$$

Equations 17a, b are easily shown to be the solutions of Equations 15c, d respectively.

These values are then inserted into Equations 15a, b with i = 2 to obtain $P_{2,0}(t)$, $P_{2,1}(t)$ and then the process is repeated until all the solutions $P_{i,k}(t)$ are obtained for $1 \le i \le M + 1$, $k = 1, 2$ $t \epsilon [0, T]$.

In general, once $P_{i,k}(t)$ are known for $1 \le i \le n-1$, $k = 1, 2$ then it is an easy matter to solve for $P_{n,k}(t)$. Let us write out the equations for i = 2:

$$\dot{P}_{2,0}(t) = -\lambda P_{2,0}(t) + P_{1,0}(t)\,\lambda_{\overline{m}} e^{-\mu t} + P_{1,1}(t)\lambda_d e^{-\mu t}$$

$$\dot{P}_{2,1}(t) = \lambda_m P_{2,0}(t) - \lambda P_{2,1}(t) + \lambda_{\overline{m}}\mu_m t\, e^{-\mu t} P_{1,0}(t) + \lambda_d \mu_m t\, e^{-\mu t} P_{1,1}(t)$$

Let

$$x = P_{2,0}(t) \;,\; y = P_{2,1}(t)$$

$$\dot{x} = -\lambda x + e^{-(\lambda+\mu)t}\lambda_{\overline{m}} + e^{-(\lambda+\mu)t}\lambda_m \lambda_d t$$

$$\dot{y} = \lambda_m x - \lambda y + P_{1,0}(t)\lambda_{\overline{m}}\mu_m t\, e^{-\mu t} + P_{1,1}(t)\,\lambda_d \mu_m t\, e^{-\mu t}$$

$$\dot{y} = \lambda_m x - \lambda y + \lambda_{\overline{m}}\mu_m t\, e^{-(\lambda+\mu)t} + e^{-(\lambda+\mu)t}\lambda_m \mu_m \lambda_d t^2$$

Thus,

$$\dot{x} = -\lambda x + e^{-(\lambda+\mu)t}(\lambda_{\overline{m}} + \lambda_m \lambda_d t) \qquad \text{Eq. 18a}$$

$$\dot{y} = \lambda_m x - \lambda y + e^{-(\lambda+\mu)t}(t\,\lambda_{\overline{m}}\mu_m + \lambda_m \mu_m \lambda_d t^2) \qquad \text{Eq. 18b}$$

subject to the initial conditions $x(0) = y(0) = 0$

A numerical procedure which recursively computes the desired solutions $x(t)$ and $y(t)$ is easily developed.

D.3. Form of the General Solution

The linear differential equation given by

$$\frac{dy}{dx} + y\,P(x) = Q(x) \qquad \text{Eq. 19a}$$

in which $P(x)$ and $Q(x)$ are functions of x, only, has the solution:

$$y(x) = e^{-\int_0^x P(t)dt} \int_0^x Q(t)\, e^{\int_0^t P(s)ds}\, dt + C \qquad \text{Eq. 19b}$$

Assuming a recursive procedure is used to evaluate $P_{i,0}(t)$, $P_{i,1}(t)$ in Equations 15a, b, c, d let us write:

$$A_i(x) = \sum_{j=1}^{i-1} e^{-\mu x}\left[1 - (1+\mu_m x)e^{-\mu_d x}\right]^{i-j-1} \left(\lambda_{\overline{m}} P_{j,0}(x) + \lambda_d P_{j,1}(x)\right) \qquad \text{Eq. 19c}$$

$$B_i(x) = \sum_{j=1}^{i-1} e^{-\mu x}\left[1 - (1+\mu_m x)e^{-\mu_d x}\right]^{i-j-1} \left(\lambda_{\overline{m}}\mu_m t P_{j,0}(x) \right. \qquad \text{Eq. 19d}$$

$$\left. + \lambda_d \mu_m t\, P_{j,1}(x)\right)$$

Then for $i \geq 2$, one obtains from Equation 19b

$$P_{i,0}(t) = e^{-\lambda t} \int_0^t e^{\lambda\tau} A_i(\tau)\, d\tau \qquad \text{Eq. 19e}$$

$$P_{i,1}(t) = e^{-\lambda t} \int_0^t e^{\lambda\tau} \left(B_i(\tau) + \lambda_m P_{i,0}(\tau)\right) d\tau$$

$$= e^{-\lambda t} \int_0^t e^{\lambda\tau}\left(B_i(\tau) + \lambda_m e^{-\lambda\tau} \int_0^\tau e^{\lambda s} A_i(s)ds\right) d\tau$$

Hence,

$$P_{i,1}(t) = e^{-\lambda t}\left[\int^{t} e^{\lambda\tau} B_i(\tau) + \lambda_m \int_0^t \int_0^\tau e^{\lambda s} A_i(s)\, ds\, d\tau\right] \qquad \text{Eq. 19f}$$

Note that Equations 19e, 19f satisfy the initial conditions that

$$P_{i,0}(0) = P_{i,1}(0) = 0 \text{ for } i \geq 2.$$

Returning to Equations 18a, 18b we find using Equations 19e, f:

$$x(t) = P_{i,0}(t) = e^{-\lambda t} \int_{(}^{t} e^{\lambda\tau}\left(\lambda_{\overline{m}} e^{-(\lambda+\mu)\tau} + \lambda_m \lambda_d \tau e^{-(\lambda+\mu)\tau}\right) d\tau$$

$$= e^{-\lambda t}\left(\frac{\lambda_{\overline{m}}}{\mu}(1-e^{-\mu t})\right) + \lambda_m\lambda_d \left(\frac{1-(1+\mu t)\, e^{-\mu t}}{\mu^2}\right)$$

$$= \left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\ d}{\mu^2}\right) e^{-\lambda t} - e^{-(\lambda+\mu)t}\left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d(1+\mu t)}{\mu^2}\right)$$

$$y(t) = e^{-\lambda t}\int_{0}^{t} e^{\lambda\tau}\left[e^{-(\lambda+\mu)\tau}\left(\tau\lambda_{\overline{m}}\mu_m + \lambda_m\mu_m\lambda_d\tau^2\right) + \lambda_m x(t)\right] d\tau$$

$$= e^{-\lambda t}\int_0^t \left\{e^{-\mu\tau}\left(\tau\lambda_{\overline{m}}\mu_m + \lambda_m\mu_m\lambda_d\tau^2\right)\right.$$

$$\left. + \lambda_m\left[\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d}{\mu^2} - e^{-\mu\tau}\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d}{\mu^2}(1+\mu\tau)\right]\right\} d\tau$$

$$= \lambda_m\left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d}{\mu^2}\right) te^{-\lambda t} + e^{-\lambda t}\int_0^t e^{-\mu\tau}(\alpha + \beta\tau + \gamma\tau^2)\, d\tau$$

where

$$\alpha = -\left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d}{\mu^2}\right), \quad \beta = \lambda_{\overline{m}}\mu_m - \frac{\lambda_m\lambda_d}{\mu}$$

and

$$\gamma = \lambda_m \mu_m \lambda_d$$

Then,

$$e^{-\lambda t} \int_0^t e^{-\mu\tau} (\alpha + \beta\tau + \gamma\tau^2)\, d\tau =$$

$$= e^{-\lambda t} \left[\frac{\alpha}{\mu} (1 - e^{-\mu t}) + \frac{\beta}{\mu^2} \left(1 - (1 + \mu t)\, e^{-\mu t}\right) \right.$$

$$\left. + \gamma \left(\frac{t^2 e^{-\mu t}}{-\mu} + \frac{2}{\mu^3} \left(1 - (1 + \mu t) e^{-\mu t}\right)\right) \right]$$

$$= e^{-\lambda t} \left(\frac{\alpha}{\mu} + \frac{\beta}{\mu^2} + \frac{2\gamma}{\mu^3} \right) - e^{-(\lambda+\mu)t} \left(\frac{\alpha}{\mu} + \frac{\beta(1+\mu t)}{\mu^2} + \gamma \left(\frac{t^2}{\mu} + \frac{2}{\mu^3} (1 + \mu t) \right) \right)$$

$$y(t) = e^{-\lambda t} \left[\frac{\alpha}{\mu} + \frac{\beta}{\mu^2} + \frac{2\gamma}{\mu^3} + \lambda_m \left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m \lambda_d}{\mu^2} \right) t \right] - \qquad \text{Eq. 19g}$$

$$- e^{-(\lambda+\mu)(t)} \left(\frac{\alpha}{\mu} + \frac{\beta}{\mu^2} (1 + \mu t) + \gamma \left(\frac{t^2}{\mu} + \frac{2}{\mu^3} (1 + \mu t) \right) \right)$$

where

$$\left.\begin{aligned} \alpha &= -\left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m \lambda_d}{\mu^2} \right) \\ \beta &= \lambda_{\overline{m}} \mu_m - \frac{\lambda_m \lambda_d}{\mu} \\ \gamma &= \lambda_m \mu_m \lambda_d \end{aligned}\right\} \qquad \text{Eqs. 19h}$$

Thus, for M = 2, S = 1, we have the solution:

$$\frac{1}{c} R_1^1 (\lambda, \mu; T) = P_{1,0}(T) + P_{1,1}(T) + P_{2,0}(T) + P_{2,1}(T)$$

Hence,

$$\frac{1}{c}R_1^1(\lambda,\mu;T) = e^{-\lambda T}(1 + \lambda_m T) + e^{-\lambda T}\left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d}{\mu^2}\right) - e^{-(\lambda+\mu)T}\left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d}{\mu^2}(1 + \mu T)\right) + e^{-\lambda T}\left(\frac{\alpha}{\mu} + \frac{\beta}{\mu^2} + \frac{2\gamma}{\mu^3} + \lambda_m\left(\frac{\lambda_{\overline{m}}}{\mu} + \frac{\lambda_m\lambda_d}{\mu^2}\right) T\right) - e^{-(\lambda+\mu)T}\left[\frac{\alpha}{\mu} + \frac{\beta}{\mu^2}(1 + \mu T) + \gamma\left(\frac{T^2}{\mu} + \frac{2}{\mu^3}(1 + \mu T)\right)\right] \quad \text{Eq. 20}$$

where α, β, γ are given by Equations 19h.

In programming the recursive solutions of Equations 15a, b, it will be important to consider the error buildup from step to step, since roundoff errors may be significant for large M. For the processors, M is moderate, (about 7), and the problem of excessive error accumulation is probably not too significant, especially since the functions $A_i(\tau)$ and $B_i(\tau)$, appearing in Equations 19e, f are positive over $[0, \infty]$.

III. E. The Multiprocessor Reliability Problem

The multiprocessor analysis treats the case of general N in $\frac{1}{c}R_S^N(\lambda, \mu; T)$, where for $N = 1$, one has the special case of simplex reliability. All the assumptions A-1 → A-5 pertaining to simplex reliability now hold for the multiprocessor analysis.

We define the corresponding birth-death process as follows: At time t the system will be said to be in state (i, k) if the module class hasn't failed in the time interval $[0, t]$ and if i active modules have experienced no faults of any kind while N-i active modules have experienced exactly 1 maskable fault each, and if k is the index of the highest numbered module in the active state. As in the simplex case, it is implicit that if the module class hasn't failed in $[0, t]$ then none of the N-i active modules mentioned above has acquired any undetectable faults in $[0, t]$.

Let

$P_{i,k}(t)$ = Prob (System is in state (i, k) at time t)

For $0 < i < N$, the following transition equations hold:

$$P_{i,k}(t+\Delta t) = P_{i,k}(t)(1 - N\,\Delta t) + P_{i+1,k}(t)(i+1)\lambda_m \Delta t + o(\Delta t) \qquad \text{Eq. 21a}$$

$$+ \sum_{\ell=N}^{k-1} P_{i+1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} (i+1)\lambda_{\overline{m}}\mu_m t e^{-\mu t}\Delta t$$

$$+ \sum_{\ell=N}^{k-1} P_{i-1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} (N-i+1)\lambda_d e^{-\mu t}\Delta t$$

$$+ \sum_{\ell=N}^{k-1} P_{i,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} \left[i\lambda_{\overline{m}}+(N-i)\lambda_d\mu_m t\right]e^{-\mu t}\Delta t$$

For $i = 0$, one has:

$$P_{0,k}(t+\Delta t) = P_{0,k}(t)\left[1-N\lambda\Delta t\right] + P_{1,k}(t)\lambda_m \Delta t + o(\Delta t) \qquad \text{Eq. 21b}$$

$$+ \sum_{\ell=N}^{k-1} P_{1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} \lambda_{\overline{m}} e^{-\mu t}\Delta t$$

$$+ \sum_{\ell=N}^{k-1} P_{o,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} N\lambda_d\mu_m t e^{-\mu t}\Delta t$$

For $i = N$, one has:

$$P_{N,k}(t+\Delta t) = P_{N,k}(t)\left[1-N\lambda\Delta t\right] + o(\Delta t) \qquad \text{Eq. 21c}$$

$$+ \sum_{\ell=N}^{k-1} P_{N-1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} \lambda_d e^{-\mu t}\Delta t$$

$$+ \sum_{\ell=N}^{k-1} P_{N,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} N\lambda_{\overline{m}} e^{-\mu t}\Delta t$$

The boundary conditions are given by:

$$P_{N,N}(0) = 1$$

$P_{i,k}(0) = 0$ for $(i, k) \neq (N, N)$, i.e. if any one of i, k is distinct from N.

The differential-difference equations obtained from these transition equations then become:

For $0 < i < N$,

$$\dot{P}_{i,k}(t) = -N\lambda P_{i,k}(t) + P_{i+1,k}(t)(i+1)\lambda_m \qquad \text{Eq. 22a}$$
$$+ \sum_{\ell=N}^{k-1} P_{i+1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} (i+1)\lambda_{\overline{m}} e^{-\mu t}$$
$$+ \sum_{\ell=N}^{k-1} P_{i-1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} (N-i+1)\lambda_d e^{-\mu t}$$
$$+ \sum_{\ell=N}^{k-1} P_{i,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} (i\lambda_{\overline{m}}+(N-i)\lambda_d\mu_m t)\, e^{-\mu t}$$

For $i = 0$,

$$\dot{P}_{0,k}(t) = N\lambda P_{0,R}(t) + \lambda m P_{1,k}(t) + o(\Delta t) \qquad \text{Eq. 22b}$$
$$+ \sum_{\ell=N}^{k-1} P_{1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} \lambda_{\overline{m}}\mu_m t\, e^{-\mu t}$$
$$+ \sum_{\ell=N}^{k-1} P_{0,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} N\lambda_d\mu_m t\, e^{-\mu t}$$

For $i = N$,

$$P_{N,k}(t) = -N\lambda P_{N,k}(t) + \sum_{l=N}^{k-1} P_{N-1,l}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} \lambda_d e^{-\mu t} \qquad \text{Eq. 22c}$$
$$+ \sum_{\ell=N}^{k-1} P_{N,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1} N\lambda_{\overline{m}} e^{-\mu t}$$

Equation 21a is derived as follows:

In order to be at state (i, k) at time $t+\Delta t$ any one of the following mutually exclusive conditions must be satisfied:

a) Either the system was in state (i, k) at time t and in the time increment Δt (i.e., the time from t to $t+\Delta t$) no faults of any kind occurred to the N active units at time t. This gives rise to the coefficient $(1 - N\lambda\Delta t)$ of $P_{i,k}(t)$, where effects only up to the first order in Δt need be considered,

or

b) The system was in state (i+1, k) at time t and at least one of the i+1 active modules, which had incurred no faults of any kind, experienced a maskable fault during the time increment Δt, this occurring with hazard rate $(i+1)\lambda_m$; this accounts for the term $P_{i+1,k}(t)\left[(i+1)\lambda_m\Delta t\right]$.

or

c) For some ℓ, $N \leq \ell < k$, the system was in state $(i+1, \ell)$ at time t, then one of the (i+1) active modules with no faults experienced a nonmaskable fault during Δt. Then, in sequencing through the next $k-\ell-1$ potential spare replacements, each such spare had either acquired at least two maskabie faults or at least one non-maskable fault during the period $[0, t]$, thus precluding its use in the system, while the k-th indexed spare had acquired at least one maskable and no other faults of any kind during this time period, $[0, t]$. Note that is is not necessary to require that any of the $k-\ell-1$ spares which were passed over as potential replacements for the removed (faulty) active unit, experience no undetectable faults during $[0, t]$, since independent of whether such faults had or had not been acquired, the module would not be used in a multiprocessor system by virtue of assumptions A-1 and A-2. The spare sequencing probability is $\left[1-(1+\mu_m t)e^{-\mu_d t}\right]^{k-\ell-1}$ while the hazard rate associated with the occurrence of at least one unmaskable fault (to the i+1 active modules which haven't had any faults is $(i+1)\lambda_{\overline{m}}$. Finally, the probability that the k-th spare has acquired exactly one maskable and no other kind of fault in $[0, t]$ is given by $\mu_m te^{-\mu t}$.

The fourth possibility is that:

d) For some ℓ, $N \leq \ell < k$, the system is in state $(i-1, \ell)$ at time t, and $k-\ell-1$ potential spares are correctly diagnosed as being non-suitable to multiprocessor operation $\left[\text{with probability } \left(1-(1+\mu_m t)e^{-\mu_d t}\right)^{k-\ell-1} \text{ as before}\right]$

while at least one of the N-i+1 active units, each having exactly one maskable fault, experiences at least one detectable fault with hazard rate $(N-i+1)\lambda_d$, while, finally, with probability $e^{-\mu t}$, the k-th spare, which is to join the i-1 active no-fault modules, has acquired no faults of any kind during $[0,t]$.

The fifth possibility is that:

e) For some ℓ, $N \leq \ell < k$, the system is in state (i,ℓ) at time t and either one of the i active zero-fault modules acquires a non-maskable fault while the k-th spare replacement has acquired none during $[0,t]$ (this accounts for the term $i\lambda_{\overline{m}}e^{-\mu t}\Delta t$) or, alternatively, at least one of the j one-maskable fault active modules has acquired a detectable fault during Δt, with hazard rate $j\lambda_d\Delta t$ and the replacement spare, of index k, has acquired exactly one maskable fault during $[0,t]$, this occurring with probability $\mu_m te^{-\mu t}$.

The remaining possibilities that could occur would be for the system to move from state (i+2, l) at time t or from state (i+3, l) at time t, etc., to state (i, k) at time t+Δt; but these effects are all of second or higher order in Δt and drop out in passing to the limit as $\Delta t \to 0$, when one formulates the corresponding differential-difference equations for the system. These terms are all subsumed within the term $o(\Delta t)$ in Equation 21.

The derivations of Equations 21b and 21c are similar except that for the case i=0, e.g., the terms under the first summation would vanish in Equation 21a, by definition. Similarly, for Equation 21c the second summation terms involving $P_{1,1}(t)$ must vanish (in Equation 21a).

Once the (unique) solution of this system of (N+1)(S+1) differential equations with prescribed boundary conditions is obtained, one may write:

$$\frac{1}{c}R_S^N(\lambda,\mu;T) = \sum_{i=0}^{N}\sum_{k=N}^{N+S} P_{i,k}(T) \qquad \text{Eq. 23}$$

Let us next treat the special case M=N (i.e., S=0).

For $o < i < N$,

$$\dot{P}_{i,N}(t) = -N\lambda P_{i,N}(t) + (i+1)\lambda_m P_{i+1,N}(t) \qquad \text{Eq. 24a}$$

$$\dot{P}_{0,N}(t) = -N\lambda P_{0,N}(t) + \lambda_m P_{1,N}(t) \qquad \text{Eq. 24b}$$

$$\dot{P}_{N,N}(t) = -N\lambda P_{N,N}(t) \qquad \text{Eq. 24c}$$

Certainly, from Equation 24c we find, using the initial conditions,

$$P_{N,N}(t) = e^{-N\lambda t}$$

By recursive solution, using Equation 19b, we find that

$$P_{i,N}(t) = \binom{N}{i} e^{-i\lambda t} e^{-(N-i)\lambda t} (\lambda_m t)^{N-i}$$

$$= \binom{N}{i} (\lambda_m t)^{N-i} e^{-N\lambda t}, \ 0 \le i \le N \qquad \text{Eq. 25}$$

We verify this by showing that $P_{i,N}(t)$, as given by Equation 25, satisfies Equation 24a (Equation 24b is a special case of Equation 24a).

In fact,

$$\dot{P}_{i,N}(t) = \binom{N}{i} e^{-Nt}\Big[(N-i)\lambda_m (\lambda_m t)^{N-i-1} - (\lambda_m t)^{N-i} N\lambda\Big]$$

$$= -N\lambda\binom{N}{i} (\lambda_m t)^{N-i} e^{-N\lambda t} + (i+1)\lambda_m \binom{N}{i}\frac{N-i}{i+1} (\lambda_m t)^{N-i-1} e^{-N\lambda t}$$

$$= -N\lambda P_{i,N}(t) + (i+1)\lambda_m \binom{N}{i+1}\left(\lambda_m t\right)^{N-(i+1)} e^{-N\lambda t}$$

$$= -N\lambda P_{i,N}(t) + (i+1)\lambda_m P_{i+1,N}(t)$$

thus verifying Equation 24a.

Finally, we have from Equation 23,

$$\frac{1}{c} R_0^N (\lambda, \mu; T) = \sum_{i=0}^{N} \binom{N}{i} (\lambda_m T)^{N-i} e^{-N\lambda T} = (1+\lambda_m T)^N e^{-N\lambda T} \qquad \text{Eq. 26}$$

The result is, of course, obvious by inspection, but is also verifies that the Birth-Death Process approach gives the correct results, i.e., it validates the internal consistency of this approach. It is clear, a priori, that $\frac{1}{c}R_0^N$ $(\lambda, \mu; T)$ is independent of μ.

In addition to solving the special case, M=N, above, the solution given points the way as to how the general case should be solved, iteratively. In fact, for M>N first solve for $P_{i,N}(t)$ and it is clear that the solution will be exactly that obtained by Equation 25. Now set k=N+1 and solve Equations 22a, b, c for $P_{i,N}(t)$. The terms under the summation sign, viz., $\sum_{l=N}^{N}$ ( ) are all known so that the theory indicated by Equation 19b points up the solution for $P_{N,N+1}(t)$. One starts with Equation 22c and solves for $P_{N,N+1}(t)$ and then works backwards (i.e., in terms of the index i) to solve for $P_{N-1,N+1}(t)$, $P_{N-2,N+1}(t)$, . . . etc. After the $P_{i,N+1}(t)$ have been solved, the procedure is repeated (again using Equation 19b) to find $\left|P_{i,N+2}(t)\right|_{i=0}^{N}$, $\left|P_{i,N+3}(t)\right|_{i=0}^{N}$, . . . $\left|P_{i,k}(t)\right|_{i=0}^{N}$, . . . and finally for $\left|P_{i,N+S}(t)\right|_{i=0}^{N}$.

At every stage of the process, one has a linear differential equation, with variable coefficients, of the first order, and the numerical analysis is easily set up in recursive fashion. As in the simplex case the problem of roundoff error must be carefully investigated as well as that of error buildup.

III. F. The Duplex and TMR Reliability Problems

The basic assumptions of simplex reliability mentioned in III. A, above, apply equally well to both the Duplex and TMR modes of operation with the following differences regarding the hazard rates relating to maskability and unmaskability as well as detection. Since both in Duplex and TMR all faults are detectable, in the first case via the use of error correcting codes and comparison of module outputs, and for TMR via voting, the class of undetectable faults, illustrated in Figure 1, does not exist in either Duplex or TMR.

Thus, the active hazard rate due to maskable faults remains λ_m as in the simplex or multiprocessor cases, while the active hazard rate due to unmaskable faults is now given by $\lambda - \lambda_m = \lambda_{\overline{m}} + \lambda_u$. Similarly, the hazard rate due to passive maskable faults is now μ_m while the hazard rate due to passive unmaskable faults is $\mu - \mu_m = \mu_{\overline{m}} + \mu_u$. The detection hazard rates are then λ and μ, for active and passive faults respectively, since all faults are detectable in either Duplex or TMR.

With these minor modifications, the Duplex or TMR reliability analyses become special cases of the multiprocessor reliability analysis by setting N=2 and N=3 in the Equations 22a, b, c, respectively.

In these equations one must, in addition, replace each occurrence of the quantities λ_m, $\lambda_{\bar{m}}$, λ_d, μ_m, μ_d by their respective counterparts λ_m, $\lambda-\lambda_m$, λ, μ_m, μ. Thus we rewrite the basic differential equations for Duplex and TMR reliability below.

III. F. 1. Duplex Reliability Differential Equations

For i = 1,

$$\dot{P}_{1,k}(t) = -2\lambda P_{1,k}(t) + P_{2,k}(t)(2\lambda_m) \qquad \text{Eq. 27a}$$

$$+\sum_{\ell=2}^{k-1} P_{2,\ell}(t)\left[1-(1+\mu_m)e^{-\mu t}\right]^{k-\ell-1} 2(\lambda-\lambda_m)e^{-\mu t}$$

$$+\sum_{\ell=2}^{k-1} P_{0,\ell}(t)\left[1-(1+\mu_m)e^{-\mu t}\right]^{k-\ell-1} 2\lambda e^{-\mu t}$$

$$+\sum_{\ell=2}^{k-1} P_{1,\ell}(t)\left[1-(1+\mu_m)e^{-\mu t}\right]^{k-\ell-1} (\lambda-\lambda_m+\lambda\mu_m t)e^{-\mu t}$$

For i = o,

$$\dot{P}_{0,k}(t) = -2\lambda P_{0,k}(t) + \lambda_m P_{1,k}(t) \qquad \text{Eq. 27b}$$

$$+\sum_{\ell=2}^{k-1} P_{1,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} (\lambda-\lambda_m)\mu_m t e^{-\mu t}$$

$$+\sum_{\ell=2}^{k-1} P_{0,\ell}(t)\left[1-(1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} 2\lambda\mu_m t e^{-\mu t}$$

For i = 2,

$$\dot{P}_{2,k}(t) = -2\lambda p_{2,k}(t) + \sum_{\ell=2}^{k-1} p_{1,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} \lambda e^{-\mu t} \qquad \text{Eq. 27c}$$

$$+ \sum_{\ell=2}^{k-1} P_{2,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} 2(\lambda-\lambda_m)e^{-\mu t}$$

The initial conditions are $P_{2,2}(0) = 1$, $P_{i,k}(0) = 0$ for i or $k \neq 2$.

III. F. 2. TMR Reliability Differential Equations

For $0 < i < 3$,

$$\dot{P}_{i,k}(t) = -3\lambda P_{i,k}(t) + P_{i+1,k}(t)(i+1)\lambda_m \qquad \text{Eq. 28a}$$

$$+ \sum_{\ell=3}^{k-1} P_{i+1,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} (i+1)(\lambda-\lambda_m)e^{-\mu t}$$

$$+ \sum_{\ell=3}^{k-1} P_{i-1,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} (3-i)\lambda e^{-\mu t}$$

$$+ \sum_{\ell=3}^{k-1} P_{i,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} (i(\lambda-\lambda_m) + (3-i)\lambda\mu_m t)e^{-\mu t}$$

For i = 0,

$$\dot{P}_{0,k}(t) = -3\lambda P_{0,k}(t) + \lambda_m P_{1,k}(t) + \qquad \text{Eq. 28b}$$

$$+ \sum_{\ell=3}^{k-1} P_{1,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} (\lambda-\lambda_m)\mu_m t e^{-\mu t}$$

$$+ \sum_{\ell=3}^{k-1} P_{0,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} 3\lambda\mu_m t e^{-\mu t}$$

For i = 3,

$$\dot{P}_{3,k}(t) = -3\lambda P_{3,k}(t) + \sum_{\ell=3}^{k-1} P_{2,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} \lambda e^{-\mu t} \qquad \text{Eq. 28c}$$

$$+ \sum_{\ell=3}^{k-1} P_{3,\ell}(t)\left[1 - (1+\mu_m t)e^{-\mu t}\right]^{k-\ell-1} 3(\lambda-\lambda_m)e^{-\mu t}$$

Bibliography

1. J.L. Bricker, "A Unified Method for Analyzing Mission Phase Reliability for Standby and Multiple Modular Redundant Computing Systems which Allows for Degraded Performance," HAC-GSG, January 1, 1972; also appeared in IEEE Trans. in Reliability, June 1973, p. 72-77.

2. W.G. Bouricius, W.C. Carter and P.R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems, ACM 1969 Conference, p. 295-305.

3. J.P. Roth, etal, "Phase II of an Architectural Study for a Self-Repairing Computer," SAMSO TR-67-106, November 1967.

4. W.G. Bouricius, W.C. Carter, J.P. Roth, P.R. Schneider, "On-Line Reliability Calculations to Achieve a Balanced Design of an Automatically Repaired Computer," NAECON, May 1967.

5. F.P. Mathur, "Reliability Modeling and Analysis of a Dynamic TMR System Utilizing Standby Spares," Proc. Seventh Annual Allerton Conference on Circuit and System Theory, October 1969, p. 243-249.

6. F.P. Mathur, "On Reliability Modeling and Analysis of Ultrareliable Fault Tolerant Digital Systems, "IEEE Transactions on Computers, November 1971, p. 1376-1382.

7. F.P. Mathur and A. Avizienis, "Reliability Analysis and Architecture of a Hybrid-Redundant Digital System: Generalized Triple Modular Redundancy with Self-Repair", Proc. 1970 Spring Joint Computer Conference, AFIPS Conference Proc., Montvale, N.J., AFIPS Press, May 1970, p. 375-383.

8. F.P. Mathur, "Reliability Modeling and Architecture of Ultra-Reliable Fault - Tolerant Digital Computers," U.C.L.A. Ph.d Thesis, 1970, Computer Science Dept., Available through University Microfilms, A Xerox Company, Ann Arbor, Mich., 1971.

9. E.J. Kletsky, "Upper Bounds on Mean Life of Self-Repairing Systems," - IEEE Trans. on Reliability and Quality Control, V.RQC-11, No. 3, Oct. 1962, p. 43-48.

10. D.S. Taylor, "A Reliability and Comparative Analysis of Two Standby System Configurations," IEE Trans. on Reliability, Vol. R-22, No. 1, April 1973, p. 13-19.

11. D.S. Taylor, "Unpowered to Powered Failure Rate Ratio: A Key Reliability Parameter", NASA-George C. Marshall Space Flight Center, Unpublished.

12. H. Wyle and G.J. Burnett, "Some Relationships between Failure Detection Probability and Computer System Reliability," AFIPS: Fall Joint Computer Conference, Nov. 1967, p. 745-756.

13. D.A. Rennels and A. Avizienis, "RMS: A Reliability Modeling System for Self-Repairing Computers", 1973 International Symposium on Fault-Tolerant Computing, FTC-3, June 20-22, 1973, Palo Alto, Calif., p. 131-135.

14. A. Avizienis and D.A. Rennels, "Fault-Tolerance Experiments with the JPL Star Computer," CompCon-72, Sixth Annual IEEE Computer Society International Conference, Digest of Papers, San Francisco, Calif., Sept. 12-14, 1972, p. 329-332.